SIAM J. COMPUT.
Vol. 33, No. 1, pp. 1–25

IMPROVED APPROXIMATION ALGORITHMS FOR THE UNCAPACITATED FACILITY LOCATION PROBLEM*

FABIÁN A. CHUDAK† AND DAVID B. SHMOYS‡

Abstract. We consider the uncapacitated facility location problem. In this problem, there is a set of locations at which facilities can be built; a fixed cost f_i is incurred if a facility is opened at location i. Furthermore, there is a set of demand locations to be serviced by the opened facilities; if the demand location j is assigned to a facility at location i, then there is an associated service cost proportional to the distance between i and j, c_{ij}. The objective is to determine which facilities to open and an assignment of demand points to the opened facilities, so as to minimize the total cost. We assume that the distance function c is symmetric and satisfies the triangle inequality. For this problem we obtain a $(1+2/e)$-approximation algorithm, where $1+2/e \approx 1.736$, which is a significant improvement on the previously known approximation guarantees.

The algorithm works by rounding an optimal fractional solution to a linear programming relaxation. Our techniques use properties of optimal solutions to the linear program, randomized rounding, as well as a generalization of the decomposition techniques of Shmoys, Tardos, and Aardal [*Proceedings of the 29th ACM Symposium on Theory of Computing*, El Paso, TX, 1997, pp. 265–274].

Key words. facility location, approximation algorithms, randomized rounding

AMS subject classifications. 90C59, 90C27

DOI. 10.1137/S0097539703405754

1. Introduction.

The study of the location of facilities to serve clients at minimum cost has been one of the most studied themes in the field of operations research (see, e.g., the textbook edited by Mirchandani and Francis [MF90]). In this paper, we focus on one of its simplest variants, *the uncapacitated facility location problem*, also known as *the simple plant location problem*, which has been extensively treated in the literature (see, e.g., the survey by Cornuéjols, Nemhauser, and Wolsey [CNW90]). This problem can be described as follows. There is a set of potential facility locations $\mathcal{F}$; building a facility at location $i \in \mathcal{F}$ has an associated nonnegative fixed cost f_i, and any open facility can provide an unlimited amount of a certain commodity. There also is a set of clients or demand points $\mathcal{D}$ that require service; client $j \in \mathcal{D}$ has a positive demand of commodity d_j that must be shipped from one of the open facilities. If a facility at location $i \in \mathcal{F}$ is used to satisfy the demand of client $j \in \mathcal{D}$, the service or transportation cost incurred per unit is proportional to the distance from i to j, c_{ij}. The goal is to determine a subset of the set of potential facility locations at which to open facilities and an assignment of clients to these facilities so as to minimize the overall total cost, that is, the fixed costs of opening the facilities plus the total service cost. We will consider only the *metric*

*Received by the editors April 15, 2002; accepted for publication (in revised form) February 18, 2003; published electronically November 14, 2003.

http://www.siam.org/journals/sicomp/33-1/40575.html

†Institute for Operations Research, Swiss Federal Institute of Technology, ETH-Zürich, Switzerland (chudak@ifor.math.ethz.ch). This research was done while the author was a graduate student at the School of Operations Research & Industrial Engineering, Cornell University, Ithaca, NY 14853. This author's research was partially supported by NSF grants DMS-9505155 and CCR-9700029 and by ONR grant N00014-96-1-00500.

‡School of Operations Research & Industrial Engineering and Department of Computer Science, Cornell University, Ithaca, NY 14853 (shmoys@cs.cornell.edu). This author's research was partially supported by NSF grants CCR-9912422, CCR-9700029, and DMS-9505155 and ONR grant N00014-96-1-00500.

variant of the problem in which the distance function c is nonnegative, symmetric, and satisfies the triangle inequality. Throughout this paper, a *ρ-approximation algorithm* is a polynomial-time algorithm that delivers a feasible solution within a factor of ρ of optimum. Our main result is a 1.736-approximation algorithm for the metric uncapacitated facility location problem.

In contrast to the uncapacitated facility location problem, Cornuéjols, Fisher, and Nemhauser [CFN77] studied the problem in which the objective is to maximize the difference between assignment and facility costs. They showed that with this objective, the problem can be thought of as a bank account location problem as follows. A company seeks to maximize its available funds by paying bills using checks drawn on banks at different locations. More precisely, if a bill is incurred in location j and is paid with a check from location i, there is delay in the clearing time that generates a profit, such as interest, c_{ij}. On the other hand, maintaining an account at location i has a fixed cost f_i. Thus the company would like to choose a subset of locations at which to open accounts so as to maximize the difference between the total profit from clearing times minus the total fixed cost of maintaining the accounts. Notice that even though the maximization and minimization problems are equivalent from the point of view of optimization, they are not equivalent from the point of view of approximation: the maximization problem can be approximated within a constant factor, whereas the minimization problem with an arbitrary distance function is as hard as the set cover problem, and thus a c-approximation algorithm with $c = o(\log |\mathcal{D}|)$ is unlikely to exist (see [Fei98] for details). Interestingly, Cornuéjols, Fisher, and Nemhauser showed that for the maximization problem, the greedy procedure that iteratively tries to open the facility that most improves the objective function yields a solution of value within a constant factor of optimum. In contrast, Hochbaum [Hoc82] showed that the greedy algorithm is an $\Theta(\log |\mathcal{D}|)$-approximation algorithm for the minimization problem with an arbitrary distance function.

By filtering a linear programming relaxation, Lin and Vitter [LV92b] also obtained an $O(\log |\mathcal{D}|)$-approximation algorithm for the uncapacitated facility location problem when the distance function c is arbitrary. In addition, they also considered the *k-median problem*, in which only k facilities can be opened, but there are no fixed costs in the objective function. They showed how to find a solution with objective function value within $(1+\epsilon)$ of optimum, but that opens $(1+1/\epsilon)O(\log |\mathcal{D}|)\, k$ facilities. Under the assumption that the distance function is a metric, they have also shown (see [LV92a]) how to find a solution of cost no more than $2(1+\epsilon)$ of optimum, opening at most $(1 + 1/\epsilon)k$ facilities. This latter result was the starting point of most recent work on metric facility location problems; although limited to the k-median problem, it essentially contains the core ideas needed to obtain a constant approximation algorithm for the metric uncapacitated facility location problem.

The metric uncapacitated facility location problem is known to be *NP*-hard (see [CNW90]). Very recently, Guha and Khuller [GK99] and Sviridenko [Svi98] have shown that it is *MAX-SNP*-hard. In fact, Guha and Khuller have also shown that the existence of a ρ-approximation algorithm for $\rho < 1.463$ implies that $NP \subseteq TIME(n^{O(\log \log n)})$ (see also Feige [Fei98]), and combined with an observation of Sviridenko [Svi98] such an algorithm would also imply that *P=NP*.

We briefly review previous work on approximation algorithms for the metric uncapacitated facility location problem. The first constant-factor approximation algorithm was given by Shmoys, Tardos, and Aardal [STA97], who presented a 3.16-approximation algorithm, based on rounding an optimal solution of a classical linear programming relaxation for the problem, due to Balinski [Bal65]. This bound

was subsequently improved by Guha and Khuller [GK99], who provided a 2.408-approximation algorithm. Guha and Khuller's algorithm requires a stronger linear programming relaxation, in which they add to the relaxation a facility budget constraint that separately bounds the total facility cost incurred. After running the algorithms of [STA97], they use a greedy procedure (as in [CFN77] and [Hoc82]) to improve the quality of the solution: iteratively, open one facility at a time if it improves the cost of the solution. However, since the optimal facility cost is unknown, they instead consider all reasonable values of the form $(1+\epsilon)^k$ for some integer k and $\epsilon > 0$. As a consequence, they must solve a weakly polynomial number of linear programs, round the optimal solution to each, and then select the best solution found. In contrast, the 1.736-approximation algorithm presented in this paper requires the solution of just one linear program, the one introduced by Balinski [Bal65], providing as a by-product further evidence of the strength of this linear programming relaxation.

In a different line of work, Korupolu, Plaxton, and Rajaraman [KPR00] showed that a simple local improvement heuristic produces a solution within a constant factor of optimum, though the best constant they obtain is 5. Very recently, Arora, Raghavan, and Rao [ARR98] have presented a quasi-polynomial approximation scheme for the case in which demand and facility points are in $\mathbb{R}^d$, where the dimension d is fixed, and the distance function is the usual Euclidean distance. If $d = 2$, then they obtain a polynomial approximation scheme. Their method is similar to the one used by Arora [Aro98] to produce approximation schemes for the traveling salesman problem. Thus, in contrast with the algorithm presented here, it appears to have only theoretical relevance due to the inefficiency of this approach.

Without loss of generality we shall assume that the set of potential facility locations $\mathcal{F}$ and the set of demand points $\mathcal{D}$ are disjoint; let $\mathcal{N} = \mathcal{F} \cup \mathcal{D}$, $n = |\mathcal{N}|$. Even though all our results hold for the case of arbitrary nonnegative demands, for the sake of simplicity of the exposition we will assume that each demand d_j is 1 ($j \in \mathcal{D}$); thus, the cost of assigning a client j to an open facility at location i is c_{ij}. The distance between any two points $k, \ell \in \mathcal{N}$ is $c_{k\ell}$. We assume that the $n \times n$ distance matrix $(c_{k\ell})$ is nonnegative, symmetric (that is, $c_{k\ell} = c_{\ell k}$ for all $k, \ell \in \mathcal{N}$), and satisfies the triangle inequality (that is, $c_{ij} \le c_{ik} + c_{kj}$ for all $i, j, k \in \mathcal{N}$). The simplest linear programming relaxation (from [Bal65]), which we will refer to as P, is as follows:

$$\text{Minimize} \quad \sum_{j\in\mathcal{D}} \sum_{i\in\mathcal{F}} c_{ij}x_{ij} + \sum_{i\in\mathcal{F}} f_i y_i$$

$$(\mathsf{P}) \qquad \text{subject to} \qquad \sum_{i\in\mathcal{F}} x_{ij} = 1 \qquad \text{for each } j \in \mathcal{D}\ , \tag{1}$$

$$x_{ij} \le y_i \qquad \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}\ , \tag{2}$$

$$x_{ij} \ge 0 \qquad \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}. \tag{3}$$

Any 0-1 feasible solution corresponds to a feasible solution to the uncapacitated facility location problem: $y_i = 1$ indicates that a facility at location $i \in \mathcal{F}$ is open, whereas $x_{ij} = 1$ means that client $j \in \mathcal{D}$ is serviced by the facility built at location $i \in \mathcal{F}$. Inequalities (1) state that each demand point $j \in \mathcal{D}$ must be assigned to some facility, whereas inequalities (2) say that clients can be assigned only to open facilities. Thus the linear program P is indeed a relaxation of the problem. Given a feasible fractional solution $(\overline{x}, \overline{y})$, we will say that $\sum_{i\in\mathcal{F}} f_i \overline{y}_i$ and $\sum_{j\in\mathcal{D}} \sum_{i\in\mathcal{F}} c_{ij}\overline{x}_{ij}$ are, respectively, its fractional facility and service cost.

Given a feasible solution to the linear programming relaxation P, the algorithm of Shmoys, Tardos, and Aardal first partitions the demand points into clusters and

then for each cluster opens exactly one facility, which services all of the points in it. In their analysis, they show that the resulting solution has the property that the total facility cost is within a constant factor of the fractional facility cost, and the total service cost is within a constant factor of the fractional service cost. The main drawback of this approach is that *most of the time* the solution is *unbalanced*, in the sense that the first constant is approximately three times smaller than the second.

One of the simplest ways to round an optimal solution (x^*, y^*) to the linear program P is to use the randomized rounding technique of Raghavan and Thompson [RT87] as proposed by Sviridenko [Svi97] for the special case in which all of the distances are 1 or 2. The 1.2785-approximation algorithm of [Svi97] essentially opens a facility at location $i \in \mathcal{F}$ with probability y_i^* and then assigns each demand point to its nearest facility. Guha and Khuller [GK99] also considered this special case and proved a matching lower bound; that is, no better guarantee is possible unless $P = NP$. Ageev and Sviridenko [AS97] have recently shown that the randomized rounding analysis for the maximum satisfiability problem of Goemans and Williamson [GW94] can be adapted to obtain improved bounds for the maximization version of the problem studied by [CFN77].

The following simple ideas enable us to develop a rounding procedure for the linear programming relaxation P with an improved performance guarantee. We explicitly exploit optimality conditions of the linear program, and, in particular, we use properties of the optimal dual solution and complementary slackness. A key element to our improvement is the use of randomized rounding in conjunction with the approach of Shmoys, Tardos, and Aardal. To understand the essence of our approach, suppose that for each location $i \in \mathcal{F}$, independently, we open a facility at i with probability y_i^*. The difficulty arises when attempting to estimate the expected service cost: the distance from a given demand point to the closest open facility might be too large. However, we could always use the routing of the algorithm of Shmoys, Tardos, and Aardal if we knew that each cluster has a facility open. Rather than opening each facility independently with probability y_i^*, we instead open *one* facility in each cluster with probability y_i^*. The precise algorithm is not much more complicated, but the most refined analysis of it is not quite so simple. Our algorithms are randomized and can be easily derandomized using the method of conditional expectations. Our main result is the following.

THEOREM 1.1. *There is a polynomial-time algorithm that rounds an optimal solution to the linear programming relaxation* P *to a feasible integer solution whose value is within* $(1 + 2/e) \approx 1.736$ *of the optimal value of the linear programming relaxation* P.

Since the optimal linear programming value is a lower bound on the integer optimal value, the theorem yields a 1.736-approximation algorithm. The running time of the algorithm is dominated by the time required to solve the linear programming relaxation P.

Since the appearance of the preliminary version of this paper [Chu98], there have been significant strides forward on research on approximation algorithms for this problem. Most notably, Jain and Vazirani [JV01] gave a primal-dual 3-approximation algorithm for this problem, which, by virtue of no longer needing to solve the linear programming relaxation, yields a substantially more efficient algorithm. The starting point of our algorithm is the graph defined by positive fractional primal assignment variables for which complementary slackness conditions are then invoked to yield tight dual constraints; the method of Jain and Vazirani first constructs a dual solution, and this serves to define an analogous graph in which the edges have the corresponding

dual constraints hold with equality.

Subsequent algorithmic results have touched on several different paradigms. Sviridenko [Svi02] has provided a more sophisticated analysis of the approach used here, also incorporating a more clever use of the so-called pipeage rounding technique. Another significant contribution of this paper is a much simpler technique for proving one of the crucial probabilistic lemmas of our paper (as well as a generalization needed for this subsequent improvement).

A number of papers have given analyses that are primal-dual in flavor. Results of Jain, Mahdian, and Saberi [JMS02] and Mahdian, Markakis, Saberi, and Vazirani [MMSV01] gave a primal-dual based analysis of greedy-style algorithms. Mettu and Plaxton [MP00] gave an algorithm that, at first consideration, does not appear to be a primal-dual algorithm at all, but, by defining "radii" for amortizing the fixed cost needed to open a facility at a particular location, is merely doing so implicitly. At this writing, the best known performance guarantee follows from an analysis of this type: Mahdian, Ye, and Zhang have given a 1.52-approximation algorithm [MYZ02].

Further work has also been done on local search algorithms; most notably, Charikar and Guha [CG99] have given a more sophisticated neighborhood structure that is amenable to analysis to yield stronger bounds than those obtained by Korupolu, Plaxton, and Rajaraman. Kolliopoulos and Rao [KR99] have also improved on the state of the art in constructing polynomial approximation schemes for these problems; most notably, they showed that a polynomial approximation scheme could be obtained in Euclidean metric spaces of constant dimension.

2. A simple 4-approximation algorithm. In this section we present a new simple 4-approximation algorithm. Even though the guarantees we will prove in the next section are substantially better, we will use most of the ideas presented here. After stating a few definitions and simple facts, we review the work of Shmoys, Tardos, and Aardal [STA97] and introduce the dual of the linear programming relaxation P and properties of primal and dual solutions that will be useful throughout the paper. First we define the *neighborhood* of a demand point $k \in \mathcal{D}$.

DEFINITION 2.1. *If $(\overline{x}, \overline{y})$ is a feasible solution to the linear programming relaxation P and $j \in \mathcal{D}$ is any demand point, the* neighborhood *of j, $\mathsf{N}(j)$, is the set of facilities that fractionally service j, that is, $\mathsf{N}(j) = \{i \in \mathcal{F} : \overline{x}_{ij} > 0\}$.*

The following fact is a simple consequence of the previous definition and inequality (1).

FACT 1. *For each demand point $j \in \mathcal{D}$, $\sum_{i \in \mathsf{N}(j)} \overline{x}_{ij} = 1$.*

The following definition was crucial for the algorithm of Shmoys, Tardos, and Aardal [STA97].

DEFINITION 2.2. *Suppose that $(\overline{x}, \overline{y})$ is a feasible solution to the linear programming relaxation P, and let $g_j \geq 0$ for each $j \in \mathcal{D}$. Then $(\overline{x}, \overline{y})$ is* g-close *if $\overline{x}_{ij} > 0$ implies that $c_{ij} \leq g_j$ $(j \in \mathcal{D}, i \in \mathcal{F})$.*

Notice that if $(\overline{x}, \overline{y})$ is g-close and $j \in \mathcal{D}$ is any demand point, all the neighbors of j, that is, the facilities that fractionally service j, are inside the ball of radius g_j centered at j. The following lemma is from [STA97].

LEMMA 2.3. *Given a feasible g-close solution $(\overline{x}, \overline{y})$, we can find, in polynomial time, a feasible integer $3g$-close solution $(\widehat{x}, \widehat{y})$ such that*

$$\sum_{i \in \mathcal{F}} f_i \widehat{y}_i \leq \sum_{i \in \mathcal{F}} f_i \overline{y}_i.$$

We briefly sketch the proof below. The algorithm can be divided into two steps:

a clustering step and a facility opening step. The clustering step works as follows (see Table 1). Let $\mathcal{S}$ be the set of demand points that have not yet been assigned to any cluster; initially, $\mathcal{S} = \mathcal{D}$. Find the unassigned demand point $j_\circ$ with smallest g_j-value and create a new cluster *centered* at $j_\circ$. Then all of the unassigned demand points that are fractionally serviced by facilities in the neighborhood of $j_\circ$ (that is, all of the demand points $k \in \mathcal{S}$ with $\mathsf{N}(k) \cap \mathsf{N}(j_\circ) \neq \varnothing$) are assigned to the cluster centered at $j_\circ$; the set $\mathcal{S}$ is updated accordingly. Repeat the procedure until all of the demand points are assigned to some cluster (i.e., $\mathcal{S} = \varnothing$). We will use $\mathcal{C}$ to denote the set of centers of the clusters.

TABLE 1
The clustering construction of Shmoys, Tardos, and Aardal.

1.	$\mathcal{S} \leftarrow \mathcal{D}, \quad \mathcal{C} \leftarrow \varnothing$
2.	while $\mathcal{S} \neq \varnothing$
3.	choose $j_\circ \in \mathcal{S}$ with smallest g_j value ($j \in \mathcal{S}$)
4.	create a new cluster $\mathcal{Q}$ centered at $j_\circ$, $\mathcal{C} \leftarrow \mathcal{C} \cup \{j_\circ\}$
5.	$\mathcal{Q} \leftarrow \{k \in \mathcal{S} : \mathsf{N}(k) \cap \mathsf{N}(j_\circ) \neq \varnothing\}$
6.	$\mathcal{S} \leftarrow \mathcal{S} - \mathcal{Q}$

The following fact follows easily from the clustering construction and the definition of neighborhood, and is essential for the success of the algorithm.

FACT 2. *Suppose that we run the clustering algorithm of Table* 1*, using any g-close solution $(\overline{x}, \overline{y})$. Then the neighborhoods of distinct centers are disjoint; that is, if j and k are centers, $j \neq k \in \mathcal{C}$, then $\mathsf{N}(j) \cap \mathsf{N}(k) = \varnothing$.*

After the clustering step, the algorithm of [STA97] opens exactly one facility per cluster. For each center $j \in \mathcal{C}$ we open the facility $i_\circ$ in the neighborhood of j, $\mathsf{N}(j)$, with smallest fixed cost f_i and assign all the demand points in the cluster of j to facility $i_\circ$. Observe that by inequalities (2) and Fact 1, $\sum_{i \in \mathsf{N}(j)} \overline{y}_i \geq 1$; thus $f_{i_\circ} \leq \sum_{i \in \mathsf{N}(j)} f_i \overline{y}_i$. Using Fact 2, the total facility cost incurred by the algorithm is never more than the total fractional facility cost $\sum_{i \in \mathcal{F}} f_i \overline{y}_i$.

Next consider any demand point $k \in \mathcal{D}$, and suppose it belongs to the cluster centered at $j_\circ$; let $\ell \in \mathsf{N}(k) \cap \mathsf{N}(j_\circ)$ be a common neighbor, and let $i_\circ$ be the open facility in the neighborhood of $j_\circ$ (see Figure 1). Then the distance from k to $i_\circ$ can be bounded by the distance from k to ℓ (which is at most g_k) plus the distance from ℓ to $j_\circ$ (which is at most $g_{j_\circ}$) plus the distance from $j_\circ$ to $i_\circ$ (which is at most $g_{j_\circ}$). Thus, the distance from k to an opened facility is at most $2g_{j_\circ} + g_k$, which is at most $3g_k$, since $j_\circ$ was the remaining demand point with minimum g-value. Hence the total service cost can be bounded by $3 \sum_{k \in \mathcal{D}} g_k$.

Shmoys, Tardos, and Aardal used the filtering technique of Lin and Vitter [LV92b] to obtain g-close solutions (as we will describe in section 5) and then applied Lemma 2.3 to obtain the first constant factor approximation algorithm for the problem. However, a simpler g-close solution is directly obtained by using the optimal solution to the dual linear program of P. More precisely, the dual of the linear program P is given by

$$\begin{aligned}
&&&\text{Maximize} && \sum_{j \in \mathcal{D}} v_j && \\
(4) \qquad &(\mathsf{D}) \qquad &&\text{subject to} && \sum_{j \in \mathcal{D}} w_{ij} \leq f_i && \text{for each } i \in \mathcal{F}, \\
(5) \qquad &&&&& v_j - w_{ij} \leq c_{ij} && \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}, \\
(6) \qquad &&&&& w_{ij} \geq 0 && \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}.
\end{aligned}$$

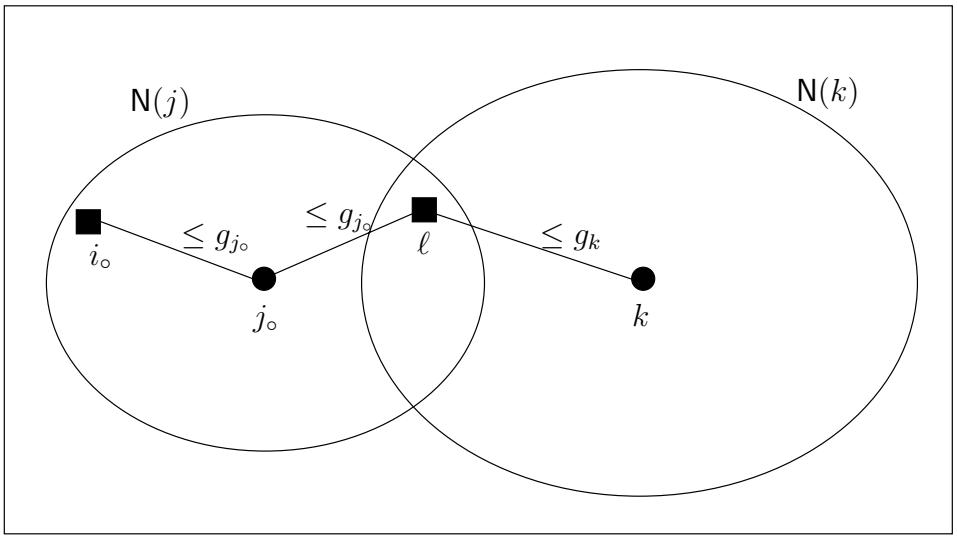


FIG. 1. *Bounding the service cost of k (4-approximation algorithm). The circles (●) are demand points, whereas the squares (■) are facility locations.*

Fix an optimal primal solution (x^*, y^*) and an optimal dual solution (v^*, w^*), and let LP^* be the optimal linear programming value. Complementary slackness gives that $x^*_{ij} > 0$ implies $v^*_j - w^*_{ij} = c_{ij}$; since $w^*_{ij} \geq 0$, we get the following lemma.

LEMMA 2.4. *If (x^*, y^*) is an optimal solution to the primal linear program P and (v^*, w^*) is an optimal solution to the dual linear program D, then (x^*, y^*) is v^*-close.*

By applying Lemma 2.3 to the optimal v^*-close solution (x^*, y^*), we obtain a feasible solution for the problem with total facility cost at most $\sum_{i\in\mathcal{F}} f_i y^*_i$ and with total service cost bounded by $3\sum_{j\in\mathcal{D}} v^*_j = 3\,\mathsf{LP}^*$. We can bound the sum of these by $4\,\mathsf{LP}^*$; thus we have a 4-approximation algorithm. Note the imbalance in bounding facility and service costs.

3. A randomized algorithm. After solving the linear program P, a very simple randomized algorithm is the following: open a facility at location $i \in \mathcal{F}$ with probability y^*_i *independently* for every $i \in \mathcal{F}$ and then assign each demand point to its closest open facility. Notice that the expected facility cost is just $\sum_{i\in\mathcal{F}} f_i y^*_i$, the same bound as in the algorithm of section 2. Focus on a demand point $k \in \mathcal{D}$. If it happens that one of its neighbors has been opened, then the service cost of k would be bounded by the optimal dual variable v^*_k. However, if we are unlucky and this is not the case (an event that, as we will see, can easily be shown to occur with probability at most $1/e \approx 0.368$, where the bound is tight), the service cost of k could be very large. On the other hand, suppose that we knew, for instance, that, for the clustering computed in section 2, k belongs to a cluster centered at j and that one of the facilities in $\mathsf{N}(j)$ has been opened. Then in this unlucky case we could bound the service cost of k using the routing cost of the 4-approximation algorithm.

Our algorithm is also based on randomized rounding, and the expected facility cost is $\sum_{i\in\mathcal{F}} f_i y^*$. However, we weaken the randomized rounding step and do *not* open facilities *independently* with probability y^*_i, but rather in a dependent way, to ensure that each cluster center has *one* of its neighboring facilities opened.

Even though the algorithms presented in this section work for any g-close feasible

solution, for the sake of simplicity of the exposition we will assume as in the end of section 2 that we have a fixed optimal primal solution (x^*, y^*) and a fixed optimal dual solution (v^*, w^*), so that (x^*, y^*) is v^*-close. It is easy to see that we can assume that $y_i^* \leq 1$ for each potential facility location $i \in \mathcal{F}$.

To motivate the following definition, fix a demand location $j \in \mathcal{D}$, and suppose without loss of generality that the neighborhood of j (that is, the facilities i for which $x_{ij}^* > 0$) is $\{1, \ldots, d\}$, with $c_{1j} \leq c_{2j} \leq \cdots \leq c_{dj}$. Then it is clear that we can assume that j is assigned "as much as possible" to facility 1, then to facility 2, and so on; that is, $x_{1j}^* = y_1^*$, $x_{2j}^* = y_2^*$, $\ldots$, $x_{d-1,j}^* = y_{d-1}^*$ (but maybe $x_{dj}^* < y_d^*$).

DEFINITION 3.1. *A feasible solution $(\overline{x}, \overline{y})$ to the linear programming relaxation* P *is* complete *if $\overline{x}_{ij} > 0$ implies that $\overline{x}_{ij} = \overline{y}_i$ for every $i \in \mathcal{F}$, $j \in \mathcal{D}$.*

Thus the optimal solution (x^*, y^*) is "almost" complete, in the sense that for every $j \in \mathcal{D}$ there is at most one $i \in \mathcal{F}$ with $0 < x_{ij}^* < y_i^*$. We point out that the notion of completeness will be helpful to highlight the main ideas of the proofs and simplify the derandomization of the algorithm, although it is not essential. Next we show that any feasible solution to P can be made complete for an equivalent instance of the problem. Recall that for a feasible solution $(\overline{x}, \overline{y})$, its fractional facility and service costs are given by, respectively, $\sum_{i \in \mathcal{F}} f_i \overline{y}_i$ and $\sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}} c_{ij} \overline{x}_{ij}$.

LEMMA 3.2. *Suppose that $(\overline{x}, \overline{y})$ is a feasible solution to the linear program* P *for a given instance of the uncapacitated facility location problem $\mathcal{I}$. Then we can find, in polynomial time, an equivalent instance $\widetilde{\mathcal{I}}$ and a complete feasible solution $(\widetilde{x}, \widetilde{y})$ to its linear programming relaxation with the same fractional facility and service costs as $(\overline{x}, \overline{y})$. The new instance $\widetilde{\mathcal{I}}$ differs only by replacing each facility location by at most $|\mathcal{D}| + 1$ copies of the same location; furthermore, if $(\overline{x}, \overline{y})$ is g-close, then so is $(\widetilde{x}, \widetilde{y})$.*

Proof. Pick any facility $i \in \mathcal{F}$ for which there is a demand point $j \in \mathcal{D}$ with $0 < \overline{x}_{ij} < \overline{y}_i$ (if there is no such facility, the original solution $(\overline{x}, \overline{y})$ is complete, and we are done). Among the demand points $j \in \mathcal{D}$ for which $\overline{x}_{ij} > 0$, let $j_\circ$ be the one with smallest $\overline{x}_{ij}$ value. Next create a new facility location i' which is an exact copy of i (i.e., the same fixed cost and in the same location), and set $\widetilde{y}_{i'} = \overline{y}_i - \overline{x}_{ij_\circ}$ and set $\widetilde{y}_i$ equal to $\overline{x}_{ij_\circ}$. Next for every $j \in \mathcal{D}$ with $\overline{x}_{ij} > 0$, set $\widetilde{x}_{ij} = \overline{x}_{ij_\circ} = \widetilde{y}_i$, and set $\widetilde{x}_{i'j} = \overline{x}_{ij} - \overline{x}_{ij_\circ}$ (which is nonnegative by the choice of $j_\circ$). All of the other components of $\overline{x}$ and $\overline{y}$ remain unchanged. Clearly, $(\widetilde{x}, \widetilde{y})$ is a feasible solution to the linear programming relaxation of the new instance; if $(\overline{x}, \overline{y})$ is g-close, so is $(\widetilde{x}, \widetilde{y})$. It is straightforward to verify that the new instance is equivalent to the old one and that the fractional facility and service costs of the solutions $(\overline{x}, \overline{y})$ and $(\widetilde{x}, \widetilde{y})$ are the same. Since the number of pairs (k, j) for which $0 < \overline{x}_{kj} < \overline{y}_k$ has decreased at least by one, and initially there can be at most $|\mathcal{D}||\mathcal{F}| \leq n^2$ such pairs, n^2 iterations suffice to construct a new instance with the desired complete solution. ∎

By Lemma 3.2, we can assume that (x^*, y^*) is complete. To understand some of the crucial points of our improved algorithm we will first consider the following RANDOMIZED ROUNDING WITH CLUSTERING. Suppose that we run the clustering procedure exactly as in Table 1, and let $\mathcal{C}$ be the set of cluster centers. We partition the facility locations into two classes, according to whether they are in the neighborhood of a cluster center or not.

DEFINITION 3.3. *The set of* central facility locations $\mathcal{L}$ *is the set of facility locations that are in the neighborhood of some cluster center, that is, $\mathcal{L} = \cup_{j \in \mathcal{C}} \mathsf{N}(j)$; the remaining set of facility locations $\mathcal{R} = \mathcal{F} - \mathcal{L}$ are* noncentral facility locations.

The algorithm opens facilities in a slightly more complicated way than the simplest randomized rounding algorithm described in the beginning of the section. First we

open exactly one central facility per cluster as follows: independently for each center $j \in \mathcal{C}$, open neighboring facility $i \in \mathsf{N}(j)$ at random with probability x^*_{ij} (recall Fact 1). Next, we independently open each noncentral facility $i \in \mathcal{R}$ with probability y^*_i. The algorithm then simply assigns each demand point to its closest open facility.

LEMMA 3.4. *For each facility location $i \in \mathcal{F}$, the probability that a facility at location i is open is y^*_i.*

Proof. If i is a noncentral facility ($i \in \mathcal{R}$), we open a facility at i with probability y^*_i. Suppose next that i is a central facility ($i \in \mathcal{L}$), and assume that $i \in \mathsf{N}(j)$ for a center $j \in \mathcal{C}$. A facility will be opened at location i only if the center j chooses it with probability x^*_{ij}; but $x^*_{ij} = y^*_i$, since (x^*, y^*) is complete. □

COROLLARY 3.5. *The expected total facility cost is $\sum_{i\in\mathcal{F}} f_i y^*_i$.*

For each demand point $k \in \mathcal{D}$, let $\overline{C}_k$ denote the fractional service cost of k, that is, $\overline{C}_k = \sum_{i\in\mathcal{F}} c_{ik} x^*_{ik}$. The expected service cost of $k \in \mathcal{D}$ is bounded in the following lemma whose proof is presented below.

LEMMA 3.6. *For each demand point $k \in \mathcal{D}$, the expected service cost of k is at most $\overline{C}_k + (3/e)v^*_k$.*

Overall, since $\sum_{k\in\mathcal{D}} v^*_k = \mathsf{LP}^*$, the expected total service cost can be bounded as follows.

COROLLARY 3.7. *The expected total service cost is at most $\sum_{k\in\mathcal{D}} \overline{C}_k + (3/e)\,\mathsf{LP}^*$.*

By combining Corollaries 3.5 and 3.7, and noting that $\sum_{k\in\mathcal{D}} \overline{C}_k + \sum_{i\in\mathcal{F}} f_i y^*_i = \mathsf{LP}^*$, we obtain the following.

THEOREM 3.8. *The expected total cost incurred by* RANDOMIZED ROUNDING WITH CLUSTERING *is at most $(1 + 3/e)\mathsf{LP}^*$.*

Proof of Lemma 3.6. Fix a demand point $k \in \mathcal{D}$. For future reference, let $j_\circ$ be the center of the cluster to which k belongs; notice that $j_\circ$ always has a neighboring facility $i_\circ$ opened (i.e., $i_\circ \in \mathsf{N}(j_\circ)$), and hence its service cost is never greater than $v^*_{j_\circ}$. To gain some intuition behind the analysis, suppose first that each center in $\mathcal{C}$ shares at most one neighbor with k; that is, $|\mathsf{N}(j) \cap \mathsf{N}(k)| \le 1$ for each center $j \in \mathcal{C}$. Each neighbor $i \in \mathsf{N}(k)$ is opened with probability $y^*_i = x^*_{ik}$ *independently* in this special case. For notational simplicity suppose that $\mathsf{N}(k) = \{1, \ldots, d\}$, with $c_{1k} \le \cdots \le c_{dk}$. Let q be the probability that none of the facilities in $\mathsf{N}(k)$ is open. Note that $q = \prod_{i=1}^{d}(1 - y^*_i) = \prod_{i=1}^{d}(1 - x^*_{ik})$. One key observation is that q is "not too big": Fact 1 combined with $1 - x \le e^{-x}$ $(x > 0)$ implies that

$$q = \prod_{i=1}^{d}(1 - x^*_{ik}) \le \prod_{i=1}^{d} e^{-x^*_{ik}} = e^{-\sum_{i=1}^{d} x^*_{ik}} = \frac{1}{e}.$$

We will bound the expected service cost of k by considering a provably worse algorithm: assign k to its closest open neighbor; if none of the neighbors of k is open, assign k to the open facility $i_\circ \in \mathsf{N}(j_\circ)$ (exactly as in section 2). If facility 1 is open, an event which occurs with probability y^*_1, the service cost of k is c_{1k}. If, on the other hand, facility 1 is closed, but facility 2 is open, an event which occurs with probability $(1 - y^*_1)y^*_2$, the service cost of k is c_{2k}, and so on. If all of the facilities in the neighborhood of k are closed, which occurs with probability q, then k is assigned to the open facility $i_\circ \in \mathsf{N}(j_\circ)$. But in this case, k is serviced by $i_\circ$, so the service cost of k is at most $2v^*_{j_\circ} + v^*_k \le 3v^*_k$ exactly as in Figure 1 (section 2); in fact, this *backup routing* gives a deterministic bound: the service cost of k is *always* no more than $3v^*_k$.

Thus the expected service cost of k is at most

$$c_{1k}\, y_1^* + c_{2k}\, y_2^*(1-y_1^*) + \cdots + c_{dk}\, y_d^*(1-y_1^*)\ldots(1-y_{d-1}^*) + 3v_k^*\, q$$
$$\leq \sum_{i=1}^{d} c_{ik} x_{ik}^* + \frac{1}{e}\, 3v_k^* = \overline{C}_k + \frac{3}{e}\, v_k^*,$$

which concludes the proof of the lemma in this special case.

Now we return to the more general case in which there are centers in $\mathcal{C}$ that can share more than one neighbor with k. We assumed that this was not the case in order to ensure that the events of opening facilities in $\mathsf{N}(k)$ were independent, but now this is no longer true for facilities $i, i' \in \mathsf{N}(k)$ that are neighbors of the same center. However, if one of i or i' is closed, the probability that the other is open increases; thus the dependencies are favorable for the analysis. The key idea of the proof is to group together those facilities that are neighbors of the same cluster center, so that the independence is retained and the proof of the special case above still works. A more rigorous analysis follows.

Let $\widehat{\mathcal{C}}$ be the subset of centers that share neighbors with k. For each center $j \in \mathcal{C}$, let $S_j = \mathsf{N}(j) \cap \mathsf{N}(k)$, and so $\widehat{\mathcal{C}} = \{j \in \mathcal{C} : S_j \neq \varnothing\}$. We have already proved the lemma when $|S_j| \leq 1$ for each center $j \in \mathcal{C}$. For each center $j \in \widehat{\mathcal{C}}$, let E_j be the event that at least one common neighbor of j and k is open (see Figure 2). To follow the proof, for each $j \in \widehat{\mathcal{C}}$, it will be convenient to think of the event of choosing facility i in S_j as a sequence of two events: first j chooses to "open" S_j with probability $p_j = \sum_{i \in S_j} x_{ik}^*$ (i.e., event E_j occurs); and then if S_j is open, j chooses facility $i \in S_j$ with probability x_{ij}^*/p_j (which is the conditional probability of opening i, given event E_j). Now let $\overline{c}_j = \sum_{i \in S_j} c_{ik} x_{ik}^*/p_j$; that is, $\overline{c}_j$ is the conditional expected distance from k to S_j, given the event E_j. For example, if $S_j = \{r, s, t\}$ are the common neighbors of j and k, the event E_j occurs when one of r, s, or t is open, $p_j = x_{rk}^* + x_{sk}^* + x_{tk}^*$, and $\overline{c}_j = c_{rk} x_{rk}^*/p_j + c_{sk} x_{sk}^*/p_j + c_{tk} x_{tk}^*/p_j$. Notice that by Fact 2, the events E_j ($j \in \widehat{\mathcal{C}}$) are independent. This completes the facility central grouping. Consider the neighbors of k that are noncentral facility locations; that is, locations $i \in \mathsf{N}(k) \cap \mathcal{R}$. For each noncentral neighbor $i \in \mathsf{N}(k) \cap \mathcal{R}$, let E_i be the event in which facility i is open, let $\overline{c}_i$ be the distance c_{ik}, and let $p_i = x_{ik}^*$. Next notice that *all* of the events E_ℓ are independent. It follows easily from the definitions that $\sum_\ell p_\ell = \sum_{i \in \mathcal{F}} \overline{x}_{ik} = 1$ and $\sum_\ell \overline{c}_\ell p_\ell = \overline{C}_k$.

Now we can argue essentially as in the simple case when $|S_j| \leq 1$ for each center $j \in \mathcal{C}$. Assume that there are d events E_ℓ, and, for notational simplicity, that they are indexed by $\ell \in \{1, \ldots, d\}$, with $\overline{c}_1 \leq \cdots \leq \overline{c}_d$. Let D be the event that none of $E_1, \ldots, E_d$ occurs; that is, D is precisely the event in which all the facilities in the neighborhood of k, $\mathsf{N}(k)$, are closed; let q be the probability of event D. Note that, as in the simple case, the service cost of k is never greater than its backup routing cost $3v_k^*$; in particular, this bound holds even conditioned on D. As before, we will analyze the expected service cost of a worse algorithm: k is assigned to the open neighboring facility with smallest $\overline{c}_\ell$; and if all the neighbors are closed, k is assigned through its backup routing to the open facility $i_\circ \in \mathsf{N}(j_\circ)$. If the event E_1 occurs (with probability p_1), the expected service cost of k is $\overline{c}_1$. If event E_1 does not occur, but event E_2 occurs (which happens with probability $(1-p_1)p_2$), the expected service cost of k is $\overline{c}_2$, and so on. If we are in the complementary space D, which occurs with probability $q = \prod_{\ell=1}^{d}(1-p_\ell)$, the service cost of k is never greater than its backup

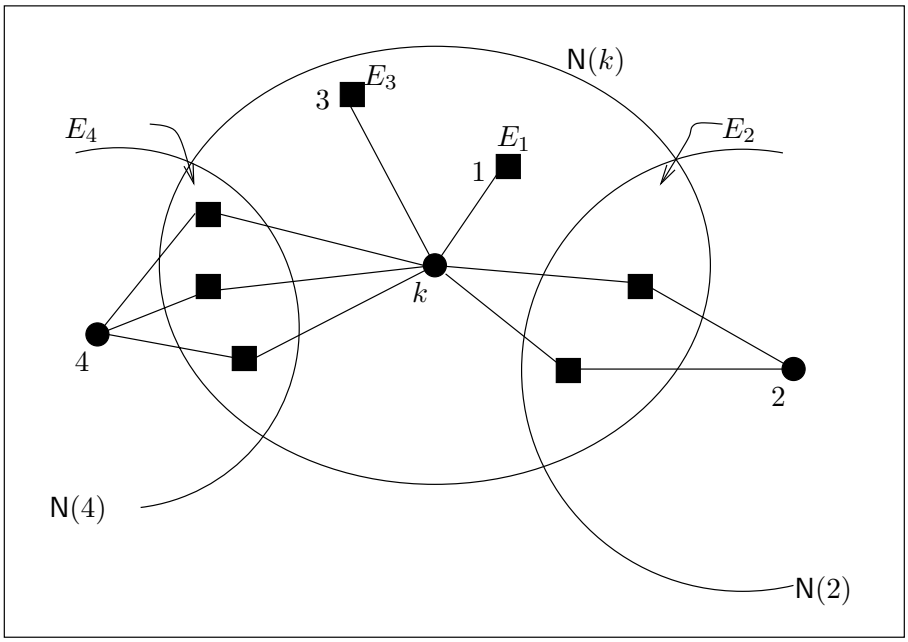


FIG. 2. *Estimating the expected service cost of k. Here the centers that share a neighbor with k are demand locations 2 and 4 ($\widehat{\mathcal{C}} = \{2, 4\}$). The neighbors of k that are noncentral locations are 1 and 3. Event E_2 (respectively, E_4) occurs when a facility in $\mathsf{N}(k) \cap \mathsf{N}(2)$ (respectively, $\mathsf{N}(k) \cap \mathsf{N}(4)$) is open, while event E_1 (respectively, E_3) occurs when facility 1 (respectively, 3) is open. Though there are dependencies among the neighbors of a fixed center, the events E_1, E_2, E_3, and E_4 are independent.*

service cost $3v_k^*$. Thus the expected service cost of k can be bounded by

$$\overline{c}_1\, p_1 + \overline{c}_2\, (1 - p_1)p_2 + \cdots + \overline{c}_d\, (1 - p_1) \ldots (1 - p_{d-1})p_d + 3v_k^*\, q. \tag{7}$$

To prove the lemma we bound the first d terms of (7) by $\overline{C}_k$, and q by $1/e$. □

Notice than even though the clustering construction is deterministic, the backup service cost of k (that is, the distance between k and the facility open in $\mathsf{N}(j_\circ)$) is a random variable B. In the proof above, we used the upper bound $B \leq 3v_k^*$. In fact, the proof of Lemma 3.6 shows that the expected service cost of k is no more than $\overline{C}_k + q\,\mathsf{E}[B|D]$, where D is the event in which no neighbor k is open, as in the proof of the lemma. As can be easily seen, the upper bound used for (7) is not tight. In fact, we can get an upper bound of $(1 - q)\,\overline{C}_k + q\,\mathsf{E}[B|D]$ as follows. First note the following simple probabilistic interpretation of the first d terms of (7). Let Z_ℓ ($\ell = 1, \ldots, d$) be independent 0-1 random variables, with $\mathsf{Prob}\{Z_\ell = 1\} = p_\ell$. Consider the set of indices for which Z_ℓ is 1, and let Z be the minimum $\overline{c}_\ell$ value in this set of indices; if all of the Z_ℓ are 0, Z is defined to be 0. Then the expected value of Z is exactly equal to the first d terms of (7). Given a set of numbers S, we will use $\min_\circ(S)$ to denote the smallest element of S if S is nonempty, and 0 if S is empty, so that $Z = \min_\circ \{\overline{c}_\ell Z_\ell : \ell = 1, \ldots, d \text{ and } Z_\ell = 1\}$. The following intuitive probability lemma, whose proof is given in section 6, provides a bound on the first d terms of (7). (A much simpler proof of this lemma, based on the Chebyshev integral inequality, was observed by Sviridenko [Svi02].)

LEMMA 3.9. *Suppose that* $0 \le \bar{c}_1 \le \cdots \le \bar{c}_d$, $p_1, \ldots, p_d > 0$, *with* $\sum_{\ell=1}^{d} p_\ell = 1$. *Let* $Z_1, \ldots, Z_d$ *be* 0-1 *independent random variables, with* $\mathsf{Prob}\{Z_\ell = 1\} = p_\ell$; *let* $\overline{C} = \sum \bar{c}_\ell p_\ell$. *Then*

$$\mathsf{E}\left[\min_{\{\ell: Z_\ell = 1\}}{}_{\circ}\ \bar{c}_\ell Z_\ell + \overline{C}\prod_{\ell=1}^{d}(1 - Z_\ell)\right] \le \overline{C}.$$

Applying the lemma to the first d terms of (7), since $\mathsf{E}[\prod_{\ell=1}^{d}(1-Z_\ell)] = \prod_{\ell=1}^{d}(1-p_\ell) = q$, we have that

$$\bar{c}_1 p_1 + \bar{c}_2(1-p_1)p_2 + \cdots + \bar{c}_d(1-p_1)\ldots(1-p_{d-1})p_d \le \overline{C}_k(1-q). \tag{8}$$

Thus we have proved the following.

LEMMA 3.10. *For each demand point* $k \in \mathcal{D}$, *the expected service cost of* k *is at most* $(1-q)\,\overline{C}_k + q\,\mathsf{E}[B|D]$.

Finally, we introduce the last idea that leads to the $(1 + 2/e)$-approximation algorithm. In Figure 1, we have bounded the distance from the center $j_\circ$ to the open facility $i_\circ$, $c_{i_\circ j_\circ}$, by $v^*_{j_\circ}$. However, $i_\circ$ is selected (by the center $j_\circ$) with probability $x^*_{i_\circ j_\circ}$, and, thus, the expected length of this leg of the routing is $\sum_{i\in\mathcal{F}} c_{ij_\circ} x^*_{ij_\circ} = \overline{C}_{j_\circ}$, which in general is smaller than the estimate $v^*_{j_\circ}$ used in the proof of Lemma 3.6. Thus, to improve our bounds, we slightly modify the clustering procedure by changing line 3 of Table 1 to

3'.	choose $j_\circ \in \mathcal{S}$ with smallest $v^*_j + \overline{C}_j$ value $(j \in \mathcal{S})$

We will call the modified algorithm RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING. Notice that Lemmas 3.4 and 3.10 are unaffected by this change. We will show that the modified rule 3' leads to the bound $\mathsf{E}[B|D] \le 2v^*_k + \overline{C}_k$, improving on the bound of $3v^*_k$ we used in the proof of Lemma 3.6.

LEMMA 3.11. *If we run* RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING, *the conditional expected backup service cost of* k, $\mathsf{E}[B|D]$, *is at most* $2v^*_k + \overline{C}_k$.

Proof. Suppose that the clustering partition assigned k to the cluster with center $j_\circ$. Deterministically, we divide the proof into two cases.

Case 1. Suppose that there is a facility $\ell \in \mathsf{N}(k) \cap \mathsf{N}(j_\circ)$ such that $c_{\ell j_\circ} \le \overline{C}_{j_\circ}$ (see Figure 3(a)). Let i be the facility in $\mathsf{N}(j_\circ)$ that was opened by $j_\circ$; notice that $c_{ij_\circ} \le v^*_{j_\circ}$ (because $(\overline{x}, \overline{y})$ is v^*-close). Then the service cost of k is at most $c_{ik} \le c_{k\ell} + c_{\ell j_\circ} + c_{j_\circ i}$, which, using again that (x^*, y^*) is v^*-close, is at most $v^*_k + c_{\ell j_\circ} + v^*_{j_\circ} \le v^*_k + \overline{C}_{j_\circ} + v^*_{j_\circ} \le \overline{C}_k + 2v^*_k$, where the last inequality follows from the fact that the center has the minimum $(\overline{C}_j + v^*_j)$ value. In this case, we have a (deterministic) bound, $B \le \overline{C}_k + 2v^*_k$.

Case 2. Assume that $c_{\ell j_\circ} > \overline{C}_{j_\circ}$ for every $\ell \in \mathsf{N}(k) \cap \mathsf{N}(j_\circ)$ (see Figure 3(b)). First note that when we do not condition on D (i.e., that no facility in $\mathsf{N}(k)$ is open), then the expected length of the edge from $j_\circ$ to the facility that $j_\circ$ has selected is $\overline{C}_{j_\circ}$. However, we are given that all of the facilities in the neighborhood of k are closed, but, in this case, all of these facilities that contribute to the expected service cost of $j_\circ$ (the facilities in $\mathsf{N}(k) \cap \mathsf{N}(j_\circ)$) are at a distance greater than the average $\overline{C}_{j_\circ}$. Thus the conditional expected service cost of $j_\circ$ is at most the unconditional expected service cost of $j_\circ$, $\overline{C}_{j_\circ}$. It follows then that if $\ell \in \mathsf{N}(k) \cap \mathsf{N}(j_\circ)$, the conditional expected service cost of k is at most $\overline{C}_{j_\circ} + c_{j_\circ \ell} + c_{\ell k} \le \overline{C}_{j_\circ} + v^*_{j_\circ} + v^*_k \le \overline{C}_k + 2v^*_k$, where again the last inequality follows from the fact that $\overline{C}_{j_\circ} + v^*_{j_\circ} \le \overline{C}_k + v^*_k$. Hence, $\mathsf{E}[B|D] \le \overline{C}_k + 2v^*_k$ in this case too. □

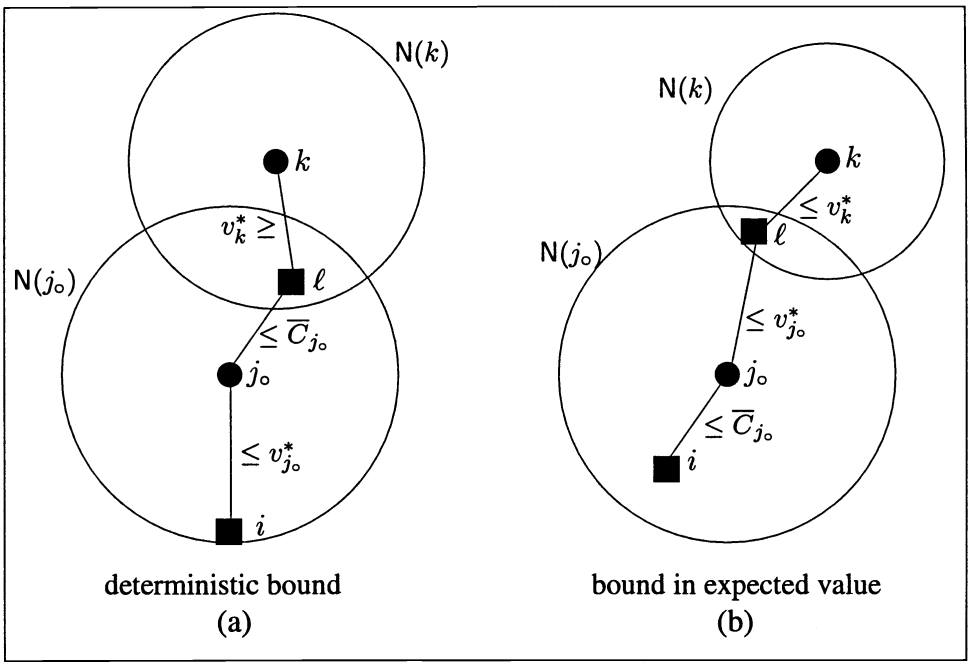


FIG. 3. *Bounding the backup service cost of k.*

Thus, using Lemmas 3.10 and 3.11, the expected service cost of k can be bounded by

$$\overline{C}_k(1-q) + q\,(2v_k^* + \overline{C}_k) = \overline{C}_k + 2q\,v_k^* \leq \overline{C}_k + \frac{2}{e}v_k^*,$$

where once again we bound q by $1/e$.

COROLLARY 3.12. *The expected total service cost of* RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING *is at most* $\sum_{k\in\mathcal{D}} \overline{C}_k + (2/e)\sum_{k\in\mathcal{D}} v_k^*$.

Combining Corollaries 3.5 and 3.12, RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING produces a feasible solution with expected cost no greater than

$$\sum_{i\in\mathcal{F}} f_i y_i^* + \sum_{k\in\mathcal{D}} \overline{C}_k + \frac{2}{e}\sum_{k\in\mathcal{D}} v_k^* = \left(1 + \frac{2}{e}\right)\mathsf{LP}^* \approx 1.736\ \mathsf{LP}^*.$$

Thus we have proved the following theorem.

THEOREM 3.13. *There is a polynomial-time randomized algorithm that finds a feasible solution to the uncapacitated facility location problem with expected cost at most* $(1 + 2/e)\ \mathsf{LP}^*$.

As a consequence of the theorem, we obtain the following corollary on the quality of the value of the linear programming relaxation of [Bal65].

COROLLARY 3.14. *The optimal value of the linear programming relaxation* P *is within a factor of* 1.736 *of the optimal cost.*

This improves on the previously best known factor of 3.16 presented in [STA97].

To finish the proof of Theorem 1.1 we will show in the next section that the algorithm of Theorem 3.13 can be derandomized using standard methods.

4. Derandomization. In this section we show how to derandomize the algorithm of Theorem 3.13. To this end, we will use the method of conditional expectations due to Erdös and Selfridge [ES73] (see also Spencer [Spe87]) that works as follows. Suppose that we have m 0-1 random variables $U_1, \ldots, U_m$ and know that $\mathsf{E}[F] = G$, where $F = g(U_1, \ldots, U_m)$ for a nonnegative real-valued function g of m variables. Our task is to determine a set of values $\overline{u}_1, \ldots, \overline{u}_m$ for the random variables $U_1, \ldots, U_m$ such that $g(\overline{u}_1, \ldots, \overline{u}_m) \leq G$. We wish to decide whether to set U_1 to 0 or 1. The expected value $\mathsf{E}[F]$ is a convex combination of the conditional expectations $\mathsf{E}[F|U_1 = 0]$ and $\mathsf{E}[F|U_1 = 1]$; more precisely,

$$\mathsf{E}[F] = \mathsf{Prob}\{U_1 = 0\}\,\mathsf{E}[F|U_1 = 0] + \mathsf{Prob}\{U_1 = 1\}\,\mathsf{E}[F|U_1 = 1].$$

We would have certainly made the right decision if the conditional expectation decreases. Thus we set $\overline{u}_1$ to 1 if $\mathsf{E}[F|U_1 = 1] \leq \mathsf{E}[F|U_1 = 0]$, and 0 otherwise. Note that $\mathsf{E}[F|U_1 = \overline{u}_1] \leq \mathsf{E}[F] = G$. The process continues inductively. Suppose that we have already decided on the values $\overline{u}_1, \ldots, \overline{u}_t \in \{0, 1\}$ so that the conditional expected value $\mathsf{E}[F|U_1 = \overline{u}_1, \ldots, U_t = \overline{u}_t]$ is at most G. Now $\mathsf{E}[F|U_1 = \overline{u}_1, \ldots, U_t = \overline{u}_t]$ is a convex combination of $\mathsf{E}[F|U_1 = \overline{u}_1, \ldots, U_t = \overline{u}_t, U_{t+1} = 0]$ and $\mathsf{E}[F|U_1 = \overline{u}_1, \ldots, U_t = \overline{u}_t, U_{t+1} = 1]$. Again we choose to set $\overline{u}_{t+1}$ to 1 if the expected value decreases, that is, if $\mathsf{E}[F|U_1 = \overline{u}_1, \ldots, U_t = \overline{u}_t, U_{t+1} = 1] \leq \mathsf{E}[F|U_1 = \overline{u}_1, \ldots, U_t = \overline{u}_t, U_{t+1} = 0]$, and 0 otherwise. Clearly, at termination, we obtain a set of 0-1 values $\overline{u}_1, \ldots, \overline{u}_m$ such that $g(\overline{u}_1, \ldots, \overline{u}_m) \leq G$ as desired. Notice that we need only to compute $2m$ conditional expected values. However, for this process to work we have to be able to compute all the intermediary conditional expectations efficiently (i.e., in polynomial time).

We first show that the analysis of RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING also provides a random variable W, a *pessimistic estimator* [Rag88], such that the cost of the solution delivered by the algorithm is at most W, and the bounds we obtained can be read off as a bound on the expected value of W, that is, $\mathsf{E}[W] \leq (1 + 2/e)\mathsf{LP}^*$. Furthermore, the expected value of W can be computed exactly. The upper bound W will let us use the method of conditional expectations to derandomize the algorithm.

The random variable W is precisely the cost of the worse algorithm we used to prove Theorem 3.13; this algorithm either assigns each demand k to the closest open neighbor (in the $\overline{c}$ sense as in the proof of Lemma 3.6) or, if none of its neighbors is open, it assigns k to the backup facility of the cluster to which k belongs. More concretely, for each potential facility location $i \in \mathcal{F}$ let U_i be the 0-1 random variable that is 1 exactly when a facility at location i is open. Notice that the random variables U_i are not independent in general, and that, since we assumed that the feasible solution $(\overline{x}, \overline{y})$ was complete, the expected value of U_i is $\overline{y}_i$. It follows immediately that the facility cost of the randomized algorithm is $\sum_{i \in \mathcal{F}} f_i U_i$.

Next we need to find an upper bound for the service costs expressed in terms of the U_i's. We fix a demand location $k \in \mathcal{D}$ and carefully revise the bounds on the expected service cost of k given in the previous section. Recall the notation used in the proof of Lemma 3.6. For each $\ell \in \{1, \ldots, d\}$, we extend the definition of S_ℓ when ℓ is a neighbor of k by just setting S_ℓ equal to $\{\ell\}$. Note that in any case $\overline{c}_\ell p_\ell = \sum_{i \in S_\ell} c_{ik} \overline{y}_i = \mathsf{E}[\sum_{i \in S_\ell} c_{ik} U_i]$. Now let $P_\ell = \sum_{i \in S_\ell} U_i$ and $T_\ell = \sum_{i \in S_\ell} c_{ik} U_i$ for $\ell = 1, \ldots, d$; and let $Q = \prod_{\ell=1}^{d} (1 - P_\ell)$. Observe that because of the dependencies, the random variables P_ℓ are 0-1 with $p_\ell = \mathsf{E}[P_\ell] = \sum_{i \in S_\ell} x^*_{ik}$. Now suppose that according to Lemma 3.11 we are in Case 1. Then the arguments given to bound the

expected service cost of k in effect say that the service cost of k is at most

$$T_1 + T_2(1-P_1) + \cdots + T_d(1-P_1)\ldots(1-P_{d-1}) + Q(v_k^* + v_{j_\circ}^* + \overline{C}_{j_\circ}); \tag{9}$$

let Z_k denote this upper bound. Since the factors of the products in each term are independent, it is straightforward to verify that the bound on the service cost of k of Theorem 3.13 implies $\mathsf{E}[Z_k] \le \overline{C}_k + (2/e)v_k^*$.

For Case 2, the service cost of k is at most

$$T_1 + T_2(1-P_1) + \cdots + T_d(1-P_1)\ldots(1-P_{d-1}) + Q\Big(v_k^* + v_{j_\circ}^* + \sum_{i\in\mathsf{N}(j_\circ)-\mathsf{N}(k)} c_{ij}U_i\Big), \tag{10}$$

which as before will be denoted by Z_k. This case is slightly more complicated than Case 1, since before we used the fact that our upper bound was deterministic. Now dependencies have to be taken into account. As in Case 1, the expected value of the first d terms can be upper bounded, exactly as in the proof of Theorem 3.13, by $(1-q)\overline{C}_k$. For the last term, notice first that the dependencies imply that $P_{j_\circ}U_i = 0$ (and thus $(1-P_{j_\circ})U_i = U_i$) for each $i \in \mathsf{N}(j_\circ) - \mathsf{N}(k)$; hence, if $Q' = \prod_{j\neq j_\circ}(1-P_j)$,

$$Q \sum_{i\in\mathsf{N}(j_\circ)-\mathsf{N}(k)} c_{ij}U_i = Q'\,(1-P_{j_\circ}) \sum_{i\in\mathsf{N}(j_\circ)-\mathsf{N}(k)} c_{ij}U_i = Q' \sum_{i\in\mathsf{N}(j_\circ)-\mathsf{N}(k)} c_{ij}U_i.$$

Now the two factors on the rightmost expression are independent, and

$$\mathsf{E}\left[\sum_{i\in\mathsf{N}(j_\circ)-\mathsf{N}(k)} c_{ij}U_i\right] \le (1-p_{j_\circ})\overline{C}_{j_\circ},$$

since, by Case 2, $c_{ij_\circ} > \overline{C}_{j_\circ}$ for all $i \in S_{j_\circ}$. Thus the expected value of the last term of (10) can be bounded by $q(v_k^* + v_{j_\circ}^* + \overline{C}_{j_\circ})$. Putting the pieces together, we have again that $\mathsf{E}[Z_k] \le \overline{C}_k + (2/e)v_k^*$ as needed. Notice also that, as in Case 1, Z_k can be written as a sum in which each term is the product of independent random variables.

Clearly, if $W = \sum_{i\in\mathcal{F}} f_iU_i + \sum_{k\in\mathcal{D}} Z_k$, the cost of the randomized algorithm can be bounded by W, and $\mathsf{E}[W] \le (1+2/e)\mathsf{LP}^*$. Since W is the cost of a worse algorithm, if we were able to find 0-1 values for the U_i's so that the corresponding value of W is at most its expected value $\mathsf{E}[W]$, we would have a feasible solution to the problem whose cost is at most $(1+2/e)\mathsf{LP}^*$, thus proving Theorem 1.1. As mentioned earlier, to find such values for the U_i's we apply the method of conditional expectations.

We need to explain how to compute the conditional expected values of W. To understand how to apply the method of conditional expectations suppose first that *all* of the U_i's are independent random variables. Now the conditional expected value $\mathsf{E}[W|U_i = 0]$ (respectively, $\mathsf{E}[W|U_i = 1]$) is easy to compute: simply replace U_i by 0 (respectively, by 1) in the expression of W and compute the expected value directly. It is also easy to see that we can substitute each U_i by $u_i \in \{0,1\}$ for $i \in \mathcal{E}$, for any subset $\mathcal{E} \subseteq \mathcal{F}$, and compute $\mathsf{E}[W|U_i = u_i \ (i \in \mathcal{E})]$. Thus, if we did not have dependencies, the derandomization of the algorithm is quite simple. However, if, for instance, U_s and U_t ($s,t \in \mathcal{F}$, $s \neq t$) are dependent, when conditioning on the event $\{U_s = 1\}$, we cannot just replace U_s by 1 in the expression of W, since the value of U_t might also be affected. This apparent difficulty can be easily overcome in our case,

since the dependencies imply that $U_t = 0$ whenever $U_s = 1$. Next we describe the derandomization process with more detail.

We first consider the noncentral facilities. If i is a noncentral facility location, $i \in \mathcal{R}$, it is easy to compute the conditional expected value of W, given that $U_i = u$ $(u = 0, 1)$: simply substitute U_i by u in the expression of W and take expected values. In the same way, we can also compute $\mathsf{E}[W|U_i = u_i \ (i \in \mathcal{E})]$, for any subset $\mathcal{E} \subseteq \mathcal{R}$, where $u_i \in \{0, 1\}$ $(i \in \mathcal{E})$. Now we can determine the values of $\overline{u}_i$ for $i \in \mathcal{R}$ applying the method of conditional expectations as described earlier. For notational simplicity, we will assume that $\mathcal{R} = \{1, \ldots, r\}$. In the first step, we set $\overline{u}_1$ equal to 1 (or, equivalently, open facility 1) only if $\mathsf{E}[W|U_1 = 1] \leq \mathsf{E}[W|U_1 = 0]$; otherwise, we set $\overline{u}_1$ equal to 0. Inductively, if we already know $\overline{u}_1, \ldots, \overline{u}_{h-1}$, in step h we set U_h to 1 if $\mathsf{E}[W|U_1 = \overline{u}_1, \ldots, U_{h-1} = \overline{u}_{h-1}, U_h = 1] \leq \mathsf{E}[W|U_1 = \overline{u}_1, \ldots, U_{h-1} = \overline{u}_{h-1}, U_h = 0]$, and to 0 otherwise. When $h = r$, we have values $\overline{u}_1, \ldots, \overline{u}_r$ such that if $\overline{W}$ is the conditional random variable $W|U_1 = \overline{u}_1, \ldots, U_r = \overline{u}_r$, then $\mathsf{E}[\overline{W}] \leq \mathsf{E}[W]$.

In what follows we find values for the remaining variables U_i, that is, decide which central facilities to open, in such a way that at termination the cost of the solution is at most $\mathsf{E}[\overline{W}]$. Fix a center $j \in \mathcal{C}$. We wish to decide which neighboring facility to open. For each neighbor $i \in \mathsf{N}(j)$ we can compute the conditional expectation of $\overline{W}$, given that U_i is 1 as follows. Since $U_i = 1$, for all the other neighbors i' of j it must be that $U_{i'} = 0$. Hence we just replace these values in the formula of $\overline{W}$ and compute the corresponding expectation. The key observation is that, since we open exactly one facility in the neighborhood of j, $\mathsf{E}[\overline{W}]$ is a convex combination of the conditional expected values $\mathsf{E}[\overline{W}|U_i = 1]$ for $i \in \mathsf{N}(j)$, that is,

$$\mathsf{E}[\overline{W}] = \sum_{i \in \mathsf{N}(j)} \mathsf{Prob}\{U_i = 1\}\, \mathsf{E}[\overline{W}|U_i = 1].$$

Thus we open the neighboring facility $i_\circ \in \mathsf{N}(j)$ for which the conditional expected value is smallest; more precisely, we set $\overline{u}_{i_\circ}$ equal to 1 and set $\overline{u}_i$ equal to 0 for $i \in \mathsf{N}(j), i \neq i_\circ$. Using now that the neighborhoods of distinct centers are disjoint, we can essentially argue as in the simple case when all the variables were independent. We repeat the process inductively; at each step we treat a new center and decide which neighboring facility to open so that the conditional expected value never increases.

Finally, notice that there are $|\mathcal{R}| + |\mathcal{C}|$ steps, and overall we need only to compute $1 + 2|\mathcal{R}| + |\mathcal{L}|$ expected values. The most expensive computation that dominates the whole derandomization process is to find, initially, the expected value of W, which takes $O(|\mathcal{D}||\mathcal{F}| \log |\mathcal{F}|)$ arithmetic operations.

5. Extensions. To motivate the results of this section, suppose that the contribution of the fractional facility cost to the optimal fractional cost is very small. If we run RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING and focus on the analysis, we would have an imbalance between the facility and service cost upper bounds. Hence, it is intuitively clear that we could afford to open facilities with higher probability to balance out the bounds. In this section, we will show how to do this in general, providing a more refined performance guarantee that depends on information related to the cost distribution of the optimal fractional solution.

A standard technique to improve the performance of randomized rounding consists of using a nontrivial mapping of the optimal fractional solution into probabilities. One of the most common approaches boosts all the probabilities by a factor of γ for a fixed parameter $\gamma > 0$. For instance, the simplest randomized rounding algorithm

would open facility $i \in \mathcal{F}$ with probability $\min\{\gamma y_i^*, 1\}$. These ideas can also be applied to our randomized algorithm in a simple fashion that we describe below.

First of all, as mentioned in section 3, the proof of Theorem 3.13 is valid for any g-close solution; that is, if $(\overline{x}, \overline{y})$ is g-close, RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING produces a feasible solution with expected cost at most

$$\sum_{i \in \mathcal{F}} f_i \overline{y}_i + \sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}} c_{ij} \overline{x}_{ij} + \frac{2}{e} \sum_{j \in \mathcal{D}} g_j.$$

In fact, it is also easy to see that the derandomization of section 4 also carries over to this more general setting.

Now suppose again that $(\overline{x}, \overline{y})$ is g-close and complete, and let $\gamma \geq 1$. We next describe the algorithm γ-RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING, which is a variant of RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING. The only difference is that now we open facilities with higher probabilities. Each noncentral facility $i \in \mathcal{R}$ is opened independently with probability $\min\{\gamma \overline{y}_i, 1\}$. As in RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING, each center $j \in \mathcal{C}$ opens one facility in its neighborhood $i \in \mathsf{N}(j)$, with probability $\overline{x}_{ij}$, in a *first phase*. Additionally, if a facility $i \in \mathsf{N}(j)$ has not been opened, it is now opened in a *second phase* with probability $\min\{\gamma \overline{y}_i - \overline{x}_{ij}, 1\} = \min\{\gamma \overline{y}_i - \overline{y}_i, 1\}$. The new algorithm has a guarantee given by the following theorem, whose proof is a minor variation of the proof of Theorem 3.13.

THEOREM 5.1. *For each $\gamma \geq 1$, the expected cost of the solution produced by* γ-RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING *is at most*

$$\gamma \sum_{i \in \mathcal{F}} f_i \overline{y}_i + \sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}} c_{ij} \overline{x}_{ij} + \frac{2}{e^\gamma} \sum_{j \in \mathcal{D}} g_j.$$

Proof. For simplicity, we will analyze a worse algorithm in which we are allowed to open more than one facility at each location $i \in \mathcal{F}$. More precisely, for each facility $i \in \mathcal{L}$, if i is in the neighborhood of center j, we open a facility at location i in the second phase with probability $\min\{\gamma \overline{y}_i - \overline{y}_i, 1\}$ independently of whether there is already an open facility at i (from the first phase); in this case, we of course take into account the extra facility cost.

First we estimate the expected total facility cost as in Lemma 3.4. If $i \in \mathcal{R}$ is a noncentral facility, i is open with probability $\min\{\gamma \overline{y}_i, 1\}$, giving a contribution to the expected total facility cost of $f_i \min\{\gamma \overline{y}_i, 1\} \leq \gamma f_i \overline{y}_i$. Now if $i \in \mathcal{L}$ is a central facility location, $i \in \mathsf{N}(j)$ for $j \in \mathcal{C}$, a facility is open at i first with probability $\overline{y}_i = \overline{x}_{ij}$, contributing $f_i \overline{y}_i$. In addition, another facility is open at i with probability $\min\{\gamma \overline{y}_i - \overline{y}_i, 1\}$, contributing an additional $f_i \min\{\gamma \overline{y}_i - \overline{y}_i, 1\}$ to the expected total facility cost. Overall, the contribution of i is $f_i \overline{y}_i + f_i \min\{\gamma \overline{y}_i - \overline{y}_i, 1\} \leq \gamma f_i \overline{y}_i$, once again. Thus, the expected total facility cost is at most $\gamma \sum_{i \in \mathcal{F}} f_i \overline{y}_i$.

Next focus on a demand point $k \in \mathcal{D}$. As in the proof of Lemma 3.6, for each center $j \in \mathcal{C}$, $S_j = \mathsf{N}(j) \cap \mathsf{N}(k)$ and $\widehat{\mathcal{C}} = \{j \in \mathcal{C} : S_j \neq \varnothing\}$. Now, for each $j \in \widehat{\mathcal{C}}$, the event E_j occurs when a facility in S_j is open in the first phase; the probability of event E_j is then $p_j = \sum_{i \in S_j} \overline{x}_{ik}$. In addition, for each central neighbor $i \in \mathsf{N}(k) \cap \mathcal{L}$, define the event E_i in which a facility is open at location i in the second phase; hence the probability of event E_i is $p_i = \min\{\gamma \overline{y}_i - \overline{y}_i, 1\}$. For each noncentral neighbor $i \in \mathsf{N}(k) \cap \mathcal{R}$, the event E_i occurs when a facility at location i is open, and the probability of event E_i is $p_i = \min\{\gamma \overline{y}_i, 1\}$. As in the proof of Lemma 3.6 and by the assumption at the beginning of the proof, all the events E_ℓ are *independent*. For each

$j \in \widehat{\mathcal{C}}$, $\overline{c}_j$ is the expected distance from k to S_j, given the event E_j and considering only the first phase. For each neighbor $i \in \mathsf{N}(k)$, $\overline{c}_i$ is simply the distance c_{ik}. Without loss of generality, we assume that there are d events E_ℓ and that they are indexed by $\ell \in \{1, \ldots, d\}$ with $\overline{c}_1 \leq \cdots \leq \overline{c}_d$. Let D be the event in which none of the events $E_1, \ldots, E_d$ occurs, and let $q = \prod_{\ell=1}^{d}(1-p_\ell)$ be the probability of event D. If B is the backup service cost of k, that is, the distance from k to the facility open during the first phase in the cluster to which k belongs (as in the proof of Lemma 3.10), following the proofs of Lemmas 3.6 and 3.10, the expected service cost of k is at most

$$\overline{c}_1\, p_1 + \overline{c}_2\, (1-p_1)p_2 + \cdots + \overline{c}_d\, (1-p_1)\ldots(1-p_{d-1})p_d + \mathsf{E}[B|D]\, q. \tag{11}$$

It is easy to see that the proof of Lemma 3.11 remains valid, so that $\mathsf{E}[B|D] \leq 2g_k + \sum_{i\in\mathcal{F}} c_{ik}\overline{x}_{ik}$. Following the proof of Theorem 3.13, we need only to show that the first d terms of (11) are at most $(1-q)\sum_{i\in\mathcal{F}} c_{ik}\overline{x}_{ik}$ and that $q \leq 1/e^\gamma$. We will use the following generalization of Lemma 3.9, whose proof is postponed until section 6. (A much simpler proof of this lemma, based on the Chebyshev integral inequality, was observed by Sviridenko [Svi02].)

LEMMA 5.2. *Let $0 \leq \overline{c}_1 \leq \cdots \leq \overline{c}_d$, and $z_1, \ldots, z_d > 0$, with $\sum_{\ell=1}^{d} z_\ell = 1$, and let $\gamma \geq 0$. Suppose that $Z_1, \ldots, Z_d$ are* 0-1 *independent random variables, with* $\mathsf{Prob}\{Z_\ell = 1\} = \min\{\gamma z_\ell, 1\}$*; let $\overline{C} = \sum \overline{c}_\ell z_\ell$. Then*

$$\mathsf{E}\left[\min_{Z_\ell=1}{}_{\circ}\, \overline{c}_\ell Z_\ell + \prod_{\ell=1}^{d}(1-Z_\ell)\overline{C}\right] \leq \overline{C}.$$

If $j \in \widehat{\mathcal{C}}$, let $z_j = p_j/\gamma$ so that $p_j = \gamma z_j = \min\{\gamma z_j, 1\}$. If $i \in \mathsf{N}(k) \cap \mathcal{R}$, let $z_i = \overline{y}_i = \overline{x}_{ik}$ so that $p_i = \min\{\gamma z_i, 1\}$. Finally, if $i \in \mathsf{N}(k) \cap \mathcal{L}$, with $i \in \mathsf{N}(j)$, $j \in \widehat{\mathcal{C}}$, let $z_i = \overline{x}_{ik} - \overline{x}_{ik}/\gamma$ so that $p_i = \min\{\gamma z_i, 1\}$. Now we have that $\sum_\ell z_\ell = 1$, and $\sum_\ell \overline{c}_\ell z_\ell = \sum_{i\in\mathcal{F}} c_{ik}\overline{x}_{ik}$. As in the proof of Lemma 3.10, using now Lemma 5.2, the first d terms of (11) can be bounded by $(1-q)\sum_{i\in\mathcal{F}} c_{ik}\overline{x}_{ik}$.

To finish the proof of the theorem notice that if p_ℓ is 1 for some ℓ, q is 0; otherwise,

$$q = \prod_{\ell=1}^{d}(1-p_\ell) = \prod_{\ell=1}^{d}(1-\gamma z_\ell) \leq \prod_{\ell=1}^{d} e^{-\gamma z_\ell} = e^{-\sum_{\ell=1}^{d}\gamma z_\ell} = \frac{1}{e^\gamma}. \qquad \square$$

Using similar arguments to those given in section 4, the algorithm of the theorem can be derandomized.

For the rest of the section let $\rho \in [0,1]$ be defined by $\rho\,\mathsf{LP}^* = \sum_{i\in\mathcal{F}} f_i y_i^*$. If ρ is either 0 or 1, there is an optimal solution to P which is integral; that is, all the y_i's are 0 or 1. When $\rho = 0$, the optimal solution sets y_i to 1 exactly for those facilities $i \in \mathcal{F}$ for which $f_i = 0$. The case when $\rho = 1$ is slightly more complicated and requires that the distance function be symmetric and satisfy the triangle inequality. First, for each facility location $i \in \mathcal{F}$, define D_i as the set of demand points that are at distance 0 from i. If $i, \ell \in \mathcal{F}$ and $j, k \in \mathcal{D}$, the inequality $c_{ij} \leq c_{ik} + c_{\ell k} + c_{\ell j}$ implies that for $i \neq i'$, either $D_i = D_{i'}$ or $D_i \cap D_{i'} = \varnothing$. Hence, we can partition the set of facilities into classes such that if i and i' belong to the same class, $D_i = D_{i'}$, and $D_i \cap D_{i'} = \varnothing$ otherwise. Now, since $\rho = 1$, the sets D_i ($i \in \mathcal{F}$) cover all the demand points. Finally, the optimal solution simply opens the cheapest facility in each class. (Notice that the fact that $D_i \cap D_{i'} = \varnothing$ for i and i' in different classes is crucial to argue optimality.) Thus we will assume that $0 < \rho < 1$. For each fixed ρ, we want to apply Theorem 5.1

to a g-close feasible solution to P with a conveniently chosen γ so as to improve the performance guarantee of γ-RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING.

First we consider the optimal solution (x^*, y^*), which is v^*-close. By Theorem 5.1, the expected cost of the solution produced by γ-RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING is at most

$$\gamma \sum_{i \in \mathcal{F}} f_i y_i^* + \sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}} c_{ij} x_{ij}^* + \frac{2}{e^\gamma} \sum_{j \in \mathcal{D}} v_j^*.$$

Since $\sum_{j \in \mathcal{D}} v_j^* = \mathsf{LP}^*$, and $\sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}} c_{ij} x_{ij}^* = (1-\rho)\,\mathsf{LP}^*$, the expected performance guarantee is at most

$$\gamma\,\rho + (1-\rho) + \frac{2}{e^\gamma}. \tag{12}$$

Now we choose $\gamma \geq 1$ so as to minimize (12), which gives

$$\gamma = \begin{cases} \ln(2/\rho) & \text{if } \rho \leq 2/e, \\ 1 & \text{if } \rho > 2/e. \end{cases}$$

In this case, we obtain a performance guarantee of

$$\begin{cases} 1 + \rho \ln(2/\rho) & \text{if } \rho \leq 2/e, \\ 1 + 2/e & \text{if } \rho > 2/e. \end{cases}$$

Figure 4 shows the performance of the new algorithm for each value of ρ. Notice that the performance improves when ρ gets closer to 0, but then there is no improvement, for instance, when ρ is close to 1. To improve the guarantees for this range of values of ρ, we will use a different g-close solution, the one proposed by Shmoys, Tardos, and Aardal [STA97], which provides better guarantees when ρ is close to 1.

In what follows, we apply Theorem 5.1 to the g-close solutions given in [STA97], which were obtained using the filtering technique of Lin and Vitter [LV92b] applied to the optimal solution (x^*, y^*). Fix $\alpha \in (0, 1]$. For a demand point $j \in \mathcal{D}$, suppose that $\mathsf{N}(j) = \{1, \ldots, d\}$, with $c_{1j} \leq \cdots \leq c_{dj}$. Let $\ell^* = \min\{\ell : 1 \leq \ell \leq d, \sum_{i=1}^{\ell} x_{ij}^* \geq \alpha\}$; then the *$\alpha$-point* of j, $c_j(\alpha)$ is $c_{\ell^* j}$. For each demand point $j \in \mathcal{D}$, let

$$\beta_j^\alpha = \sum_{i : c_{ij} \leq c_j(\alpha)} x_{ij}^*.$$

The filtered solution (x^α, y^α) is defined in a simple way to ensure g-closeness, for $g_j = c_j(\alpha)$ $(j \in \mathcal{D})$, as follows: if $j \in \mathcal{D}$, $i \in \mathcal{F}$,

$$x_{ij}^\alpha = \begin{cases} \dfrac{x_{ij}^*}{\beta_j^\alpha} & \text{if } c_{ij} \leq c_j(\alpha), \\ 0 & \text{otherwise,} \end{cases}$$

and $y_i^\alpha = \min\{1, y_i^*/\alpha\}$ for $i \in \mathcal{F}$. It is easy to verify that (x^α, y^α) is a feasible primal solution (because $\beta_j^\alpha \geq \alpha$) and that (x^α, y^α) is $(c_j(\alpha))$-close. The total fractional facility cost of the filtered solution is $\sum_{i \in F} f_i y_i^\alpha$, and it is clearly bounded by $\sum_{i \in F} f_i y_i^*/\alpha$. Next define $\tau(\alpha) = \sum_{j \in \mathcal{D}} c_j(\alpha) / \sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}} c_{ij} x_{ij}^*$. The following lemma follows from Lemma 10 of [STA97] and was observed in [GK99].

LEMMA 5.3. *The function $\tau(\alpha)$ satisfies the following expression:*

$$\int_0^1 \tau(\alpha)\ \mathrm{d}\alpha = 1.$$

Proof. It was shown in [STA97] that $\int_0^1 c_j(\alpha)\ \mathrm{d}\alpha = \sum_{i\in\mathcal{F}} c_{ij}x^*_{ij}$, from which the lemma follows at once according to our definitions. □

Also note that $\tau(\alpha)$ is a left continuous step function that has at most $O(|\mathcal{D}||\mathcal{F}|)$ break points. Since for each demand point $k \in \mathcal{D}$, the fractional service cost of the filtered solution is bounded by the fractional transportation cost, that is,

$$\sum_{i\in\mathcal{F}} c_{ik}x^\alpha_{ik} = \sum_{\{i:c_{ik}\le c_k(\alpha)\}} c_{ik}\frac{x^*_{ik}}{\beta^\alpha_j} \le \sum_{i\in\mathcal{F}} c_{ik}x^*_{ik},$$

the previous theorem implies that the expected total cost of the solution of γ-RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING when applied to the $(c_j(\alpha))$-close solution (x^α, y^α) is at most

$$\gamma\sum_{i\in F} f_i\frac{y^*_i}{\alpha} + \sum_{j\in\mathcal{D}}\sum_{i\in\mathcal{F}} c_{ij}x^*_{ij} + \frac{2}{e^\gamma}\sum_{j\in D} c_j(\alpha),$$

or an expected performance guarantee of

$$\gamma\frac{\rho}{\alpha} + 1 - \rho + \frac{2}{e^\gamma}(1-\rho)\tau(\alpha). \tag{13}$$

To find the best possible guarantee we will use binary search on a target guarantee and Lemma 5.3 in a way similar to that used in [GK99] as follows. Fix $c > 1$, and suppose that γ-RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING does not produce a c-approximation algorithm for every $\alpha \in (0,1)$, $\gamma \ge 1$. Then

$$\tau(\alpha) > \frac{e^\gamma}{2}\frac{(c-1+\rho)\alpha - \gamma\rho}{(1-\rho)\alpha}. \tag{14}$$

We are interested only in values of α for which the expression on the right is nonnegative; hence we will assume that $\alpha \ge \gamma\rho/(c-1+\rho)$. For fixed α, the value of $\gamma \ge 1$ that makes the right-hand side of (14) greatest is given by

$$\gamma(\alpha) = \begin{cases} \dfrac{(c-1+\rho)\alpha - \rho}{\rho} & \text{if } 2\rho/(c-1+\rho) \le \alpha \le 1, \\ 1 & \text{if } \rho/(c-1+\rho) \le \alpha \le 2\rho/(c-1+\rho). \end{cases}$$

Thus, from (14), we obtain a lower bound for $\tau(\alpha)$, $LB_c(\alpha)$, given by

$$LB_c(\alpha) = \begin{cases} \dfrac{\rho}{2}\dfrac{\exp\{[(c-1+\rho)\alpha-\rho]/\rho\}}{(1-\rho)\alpha} & \text{if } 2\rho/(c-1+\rho) \le \alpha \le 1, \\ \dfrac{e}{2}\dfrac{(c-1+\rho)\alpha-\rho}{(1-\rho)\alpha} & \text{if } \rho/(c-1+\rho) \le \alpha \le 2\rho/(c-1+\rho), \\ 0 & \text{if } 0 \le \alpha \le \rho/(c-1+\rho). \end{cases}$$

Note that $LB_c(\alpha)$ is a continuous and increasing function of α (since c and ρ are fixed). Now since $\int_0^1 \tau(\alpha)\ \mathrm{d}\alpha = 1$, and $\tau(\alpha) > LB_c(\alpha)$, if we can show that $\int_0^1 LB_c(\alpha)\ \mathrm{d}\alpha > 1$,

we have a contradiction, and, thus, the performance guarantee of the algorithm is no greater than c. Using bisection search we can find the smallest c, say $c_\circ$, within a small error tolerance (say 0.0001) for which $\int_0^1 LB_{c_\circ}(\alpha)\ \mathrm{d}\alpha > 1$, and hence show that the algorithm has a performance guarantee $c_\circ$. It is clear that the bisection search takes constant time. We finally address the issue of how to find the parameters α and γ to obtain a $c_\circ$-approximation algorithm. We know that

$$\int_0^1 \tau(\alpha)\ \mathrm{d}\alpha = 1 < \int_0^1 LB_{c_\circ}(\alpha)\ \mathrm{d}\alpha; \tag{15}$$

hence there must be a value of α, say $\alpha_\circ$, such that

$$\tau(\alpha_\circ) \le LB_{c_\circ}(\alpha_\circ). \tag{16}$$

This implies that when $\gamma = \gamma(\alpha_\circ)$, γ-RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING produces a solution with expected cost within a factor of $c_\circ$ of optimum. Thus we need only to argue how to find such an $\alpha_\circ$ in polynomial time.

Now we know that $\tau(\alpha)$ is a left continuous step function with at most $O(|\mathcal{D}|^2)$ break points. Hence we can find a partition of the interval $(0,1] = (0,s_1] \cup (s_1,s_2] \cup \cdots \cup (s_k,1]$, with $k = O(|\mathcal{D}|^2)$, such that $\tau(\alpha)$ is constant in each subinterval. Suppose that $\tau(\alpha) = z$ in the subinterval $(s_i, s_{i+1}]$. If $z > LB_{c_\circ}(s_{i+1})$, since $LB_{c_\circ}$ is increasing, the inequality must hold for the whole subinterval. Thus if for all the right endpoints of the subintervals inequality (16) does not hold, $\tau(\alpha) > LB_{c_\circ}(\alpha)$ for each $\alpha \in (0,1]$, contradicting (15). This gives a simple way to find a $c_\circ$-approximation algorithm. Figure 4 shows the performance guarantees obtained using this algorithm.

We conclude the section by pointing out that there is a minor improvement on the best guarantee we can achieve using the algorithms γ-RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING. Indeed, a careful computation gives a slightly improved overall performance guarantee of $1.73352 < 1.73576 \approx 1 + 2/e$.

6. Proof of Lemmas 3.9 and 5.2. Notice that Lemma 3.9 follows from Lemma 5.2 by simply taking $\gamma = 1$. Thus we need only to prove Lemma 5.2. As noted above, Sviridenko [Svi02] (in the proof and discussion around his Lemmas 4 and 5) observed that a much simpler version of this proof can be derived from the Chebyshev integral inequality, and he cited particular variants in the text of Hardy, Littlewood, and Pólya [HLP52].

The key idea of the proof is to observe that the expected value decreases as a function of γ; thus, since when γ is 0 the expected value is exactly $\overline{C}$, we are done.

For $\gamma \ge 0$, let $p(\gamma) := \mathsf{E}[\min_{\circ\{i:Z_i=1\}} \overline{c}_i Z_i + \prod_{i=1}^d (1-Z_i)\overline{C}]$. Let $\overline{\gamma} = \min_i(1/z_i)$. We first prove the lemma for $\gamma \le \overline{\gamma}$, so that $\gamma z_i < 1$, if $\gamma < \overline{\gamma}$. In this case, $p(\gamma)$ can be computed as

$$\overline{c}_1\gamma z_1 + \overline{c}_2\gamma z_2(1-\gamma z_1) + \cdots + \overline{c}_d\gamma z_d(1-\gamma z_1)\cdots(1-\gamma z_{d-1}) + \overline{C}\prod_{\ell=1}^{d}(1-\gamma z_\ell).$$

Notice that $p(\gamma)$ is a polynomial on γ and that $p(0) = \overline{C}$. Hence, to prove the lemma in this case it is enough to show that $p'(\gamma) \le 0$ for $\gamma \in (0, \overline{\gamma})$.

Next define for each $\ell = 0, 1, \ldots, d$

$$F_\ell(\gamma) = \begin{cases} 1 & \text{if } \ell = 0, \\ (1-\gamma z_\ell)F_{\ell-1}(\gamma) & \text{if } \ell \ge 1. \end{cases}$$

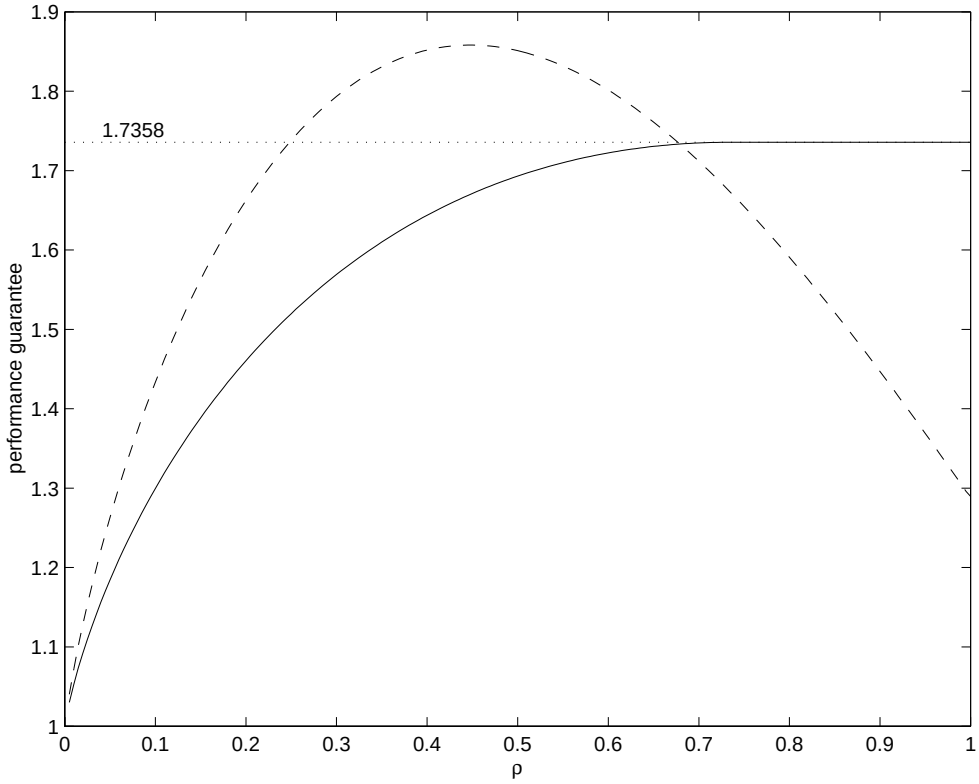


FIG. 4. *Performance guarantees of* γ-RANDOMIZED ROUNDING WITH IMPROVED CLUSTERING *as a function of* ρ. *The solid line corresponds to the algorithm that uses the optimal* v^**-close solution* (x^*, y^*), *while the dashed line corresponds to the algorithm run with the filtered solution of* [STA97].

Thus we can write

$$\begin{aligned} p(\gamma) &= \overline{c}_1\gamma z_1 F_0(\gamma) + \overline{c}_2\gamma z_2 F_1(\gamma) + \cdots + \overline{c}_d\gamma z_d F_{d-1}(\gamma) + \overline{C}F_d(\gamma) \\ &= \sum_{\ell=1}^{d} \overline{c}_\ell\gamma z_\ell F_{\ell-1}(\gamma) + F_d(\gamma)\sum_{\ell=1}^{d}\overline{c}_\ell z_\ell \\ &= \sum_{\ell=1}^{d} \overline{c}_\ell z_\ell \left[\gamma F_{\ell-1}(\gamma) + F_d(\gamma)\right]. \end{aligned}$$

Then

$$p'(\gamma) = \sum_{\ell=1}^{d} \overline{c}_\ell z_\ell \left[F_{\ell-1}(\gamma) + \gamma F'_{\ell-1}(\gamma) + F'_d(\gamma)\right].$$

For $\ell = 1, \ldots, d$, let $\lambda_\ell = F_{\ell-1}(\gamma) + \gamma F'_{\ell-1}(\gamma) + F'_d(\gamma)$ so that $p'(\gamma) = \sum_{\ell=1}^{d} \overline{c}_\ell z_\ell \lambda_\ell$. Next note that

$$F'_\ell(\gamma) = \begin{cases} 0 & \text{if } \ell = 0, \\ -z_\ell F_{\ell-1}(\gamma) + (1-\gamma z_\ell)F'_{\ell-1}(\gamma) & \text{if } \ell \geq 1. \end{cases}$$

It is easy to check by induction that $F'_\ell(\gamma) \le 0$, $\ell = 0, \dots, d$. To prove the lemma in this case, we will use the following two claims.

CLAIM 1. *The following equality holds:* $\sum_{\ell=1}^{d} z_\ell \lambda_\ell = 0$.

CLAIM 2. *There is an index* $\ell_\circ$, $0 \le \ell_\circ \le d$, *such that* $\lambda_1, \dots, \lambda_{\ell_\circ - 1} \ge 0$, *and* $\lambda_{\ell_\circ}, \dots, \lambda_d \le 0$.

To prove that $p'(\gamma) \le 0$, take $\ell_\circ$ as in Claim 2; then use Claim 1 and the order of the $\overline{c}_\ell$'s:

$$\begin{aligned} p'(\gamma) &= \sum_{\ell=1}^{\ell_\circ - 1} \overline{c}_\ell \lambda_\ell z_\ell + \sum_{\ell=\ell_\circ}^{d} \overline{c}_\ell \lambda_\ell z_\ell \\ &\le \overline{c}_{\ell_\circ} \left(\sum_{\ell=1}^{\ell_\circ - 1} \lambda_\ell z_\ell \right) + \overline{c}_{\ell_\circ} \left(\sum_{\ell=\ell_\circ}^{d} \lambda_\ell z_\ell \right) \\ &= \overline{c}_{\ell_\circ} \sum_{\ell=1}^{d} \lambda_\ell z_\ell \\ &= 0. \end{aligned}$$

We complete the proof of the lemma for $\gamma \in [0, \overline{\gamma}]$ by proving the claims.

Proof of Claim 1. Note first that for $\ell = 1, \dots, d$

$$-z_\ell F_{\ell-1}(\gamma) - \gamma z_\ell F'_{\ell-1}(\gamma) = F'_\ell(\gamma) - F'_{\ell-1}(\gamma).$$

Thus

$$\sum_{\ell=1}^{d} [-z_\ell F_{\ell-1}(\gamma) - \gamma z_\ell F'_{\ell-1}(\gamma)] = F'_d(\gamma) - F'_0(\gamma) = F'_d(\gamma),$$

which implies that $\sum_\ell z_\ell \lambda_\ell = 0$, since $\sum_\ell z_\ell = 1$. □

Proof of Claim 2. If $\lambda_\ell \ge 0$ (or $\lambda_\ell \le 0$) for all ℓ, using Claim 1, $\lambda_\ell = 0$ for all ℓ, and any index works. Hence we can assume that at least one $\lambda_\ell < 0$ and at least one $\lambda_\ell > 0$. Suppose the claim does not hold. Then there must exist an index ℓ, $1 \le \ell \le d$, such that $\lambda_\ell \le 0$, but $\lambda_{\ell+1} > 0$. In particular,

$$\lambda_{\ell+1} = F_\ell(\gamma) + \gamma F'_\ell(\gamma) + F'_d(\gamma) > 0.$$

Thus,

$$\begin{aligned} \lambda_{\ell+1} - F'_d(\gamma) &= F_\ell(\gamma) + \gamma F'_\ell(\gamma) \\ &= (1 - \gamma z_\ell) F_{\ell-1}(\gamma) + (1 - \gamma z_\ell) \gamma F'_{\ell-1}(\gamma) - \gamma z_\ell F_{\ell-1}(\gamma) \\ &= (1 - \gamma z_\ell)[F_{\ell-1}(\gamma) + \gamma F'_{\ell-1}(\gamma)] - \gamma z_\ell F_{\ell-1}(\gamma) \\ &> -F'_d(\gamma) \\ &\ge 0. \end{aligned}$$

Let $t := F_{\ell-1}(\gamma) + \gamma F'_{\ell-1}(\gamma)$. Since $(1 - \gamma z_\ell) > 0$ and $-\gamma z_\ell F_{\ell-1}(\gamma) < 0$, it must be the case that $t > 0$. Since $\lambda_\ell \le 0$, $t \le -F'_d(\gamma)$. Thus,

$$0 < t \le -F'_d(\gamma) < \lambda_{\ell+1} - F'_d(\gamma) = (1 - \gamma z_\ell) t - \gamma z_\ell F_{\ell-1}(\gamma) < (1 - \gamma z_\ell) t,$$

which is impossible since $(1 - \gamma z_\ell) < 1$. □

Next suppose that $\gamma \geq \overline{\gamma}$, and let $\ell_\circ$ be the smallest index for which $\min\{\gamma z_\ell, 1\}$ is 1. Now we have that

$$p(\gamma) = \overline{c}_1\gamma z_1 + \overline{c}_2\gamma z_2(1-\gamma z_1) + \cdots + \overline{c}_{\ell_\circ}(1-\gamma z_1)\ldots(1-\gamma z_{\ell_\circ-1}).$$

We reduce this case to the previous one as follows. Let $\alpha = \sum_{\ell=1}^{\ell_\circ-1} z_\ell$, $\gamma' = \gamma\alpha$, and let $z'_\ell = z_\ell/\alpha$ for $\ell = 1, \ldots, \ell_\circ - 1$; note that $\sum_{\ell=1}^{\ell_\circ-1} z'_\ell = 1$. We have then that

$$p(\gamma) = \overline{c}_1\gamma' z'_1 + \overline{c}_2\gamma' z'_2(1-\gamma' z'_1) + \cdots + \overline{c}_{\ell_\circ}(1-\gamma' z'_1)\ldots(1-\gamma' z'_{\ell_\circ-1}). \tag{17}$$

Notice that $\gamma' z'_\ell < 1$ for each $\ell = 1, \ldots, \ell_\circ - 1$. If $\overline{C}' = \sum_{\ell=1}^{\ell_\circ-1} \overline{c}_\ell z'_\ell$, and $q = \prod_{\ell=1}^{\ell_\circ-1}(1-\gamma' z'_\ell)$, we can apply the previous case to conclude that the first $\ell_\circ - 1$ terms of (17) can be bounded by $(1-q)\overline{C}'$. Thus,

$$p(\gamma) \leq (1-q)\overline{C}' + \overline{c}_{\ell_\circ} q. \tag{18}$$

Suppose that $q \leq 1-\alpha$. Then, since $\overline{C}' \leq \overline{c}_{\ell_\circ}$ by the ordering of the $\overline{c}$'s, from (18)

$$p(\gamma) \leq \alpha\overline{C}' + (1-\alpha)\overline{c}_{\ell_\circ} = \sum_{\ell=1}^{\ell_\circ-1} \overline{c}_\ell z_\ell + (1-\alpha)\overline{c}_{\ell_\circ} \leq \sum_{\ell=1}^{d} \overline{c}_\ell z_\ell = \overline{C},$$

and the lemma follows. Hence, to conclude the proof, we need only to argue that $q \leq 1-\alpha$. First note that since $1 - x \leq e^{-x}$ $(x \geq 0)$,

$$q = \prod_{\ell=1}^{\ell_\circ-1}(1-\gamma' z'_\ell) \leq \prod_{\ell=1}^{\ell_\circ-1} e^{-\gamma' z'_\ell} = e^{-\gamma'\sum_{\ell=1}^{\ell_\circ-1} z'_\ell} = e^{-\gamma'} = e^{-\gamma\alpha}.$$

Now, since $\gamma z_{\ell_\circ} \geq 1$, it must be that $\gamma \geq 1/(1-\alpha)$. Finally,

$$q \leq e^{-\gamma\alpha} \leq e^{-\alpha/(1-\alpha)} \leq 1-\alpha,$$

where the last inequality follows from $1 - \ln(y) \leq 1/y$ for $y \in (0,1)$ by setting $y = 1-\alpha$.

REFERENCES

[Aro98] S. Arora, *Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems*, J. ACM, 45 (1998), pp. 753–782.

[ARR98] S. Arora, P. Raghavan, and S. Rao, *Approximation schemes for Euclidean k-medians and related problems*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 106–113.

[AS97] A. A. Ageev and M. I. Sviridenko, *An Approximation Algorithm for the Uncapacitated Facility Location Problem*, manuscript, 1997.

[Bal65] M. L. Balinski, *Integer programming: Methods, uses, computation*, Management Sci., 12 (1965), pp. 253–313.

[CFN77] G. Cornuéjols, M. L. Fisher, and G. L. Nemhauser, *Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms*, Management Sci., 23 (1977), pp. 789–810.

[CG99] M. Charikar and S. Guha, *Improved combinatorial algorithms for the facility location and k-median problems*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, NY, 1999, pp. 378–388.

[Chu98] F. A. Chudak, *Improved approximation algorithms for uncapacitated facility location*, in Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 1412, R. E. Bixby, E. A. Boyd, and R. Z. Rios-Mercado, eds., Springer, Berlin, 1998, pp. 180–194.

[CNW90] G. Cornuéjols, G. L. Nemhauser, and L. A. Wolsey, *The uncapacitated facility location problem*, in Discrete Location Theory, P. Mirchandani and R. Francis, eds., John Wiley and Sons, New York, 1990, pp. 119–171.

[ES73] P. Erdös and J. L. Selfridge, *On a combinatorial game*, J. Combinatorial Theory Ser. A, 14 (1973), pp. 298–301.

[Fei98] U. Feige, *A threshold of* ln *n for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.

[GK99] S. Guha and S. Khuller, *Greedy strikes back: Improved facility location algorithms*, J. Algorithms, 31 (1999), pp. 228–248.

[GW94] M. X. Goemans and D. P. Williamson, *New* 3/4*-approximation algorithms for the maximum satisfiability problem*, SIAM J. Discrete Math., 7 (1994), pp. 656–666.

[HLP52] G. H. Hardy, J. E. Littlewood, and G. Pólya, *Inequalities*, Cambridge University Press, Cambridge, UK, 1952.

[Hoc82] D. S. Hochbaum, *Heuristics for the fixed cost median problem*, Math. Programming, 22 (1982), pp. 148–162.

[JMS02] K. Jain, M. Mahdian, and A. Saberi, *A new greedy approach for facility location problems*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, Montreal, QB, Canada, pp. 731–740.

[JV01] K. Jain and V. V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.

[KR99] S. G. Kolliopoulos and S. Rao, *A nearly linear-time approximation scheme for the Euclidean* κ*-median problem*, in Algorithms—ESA '99, Lecture Notes in Comput. Sci. 1643, J. Nesetril, ed., Springer, Berlin, 1999, pp. 378–389.

[KPR00] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, *Analysis of a local search heuristic for facility location problems*, J. Algorithms, 37 (2000), pp. 146–188.

[LV92a] J. H. Lin and J. S. Vitter, *Approximation algorithms for geometric median problems*, Inform. Process. Lett., 44 (1992), pp. 245–249.

[LV92b] J. H. Lin and J. S. Vitter, ϵ*-approximation with minimum packing constraint violation*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, Victoria, BC, Canada, pp. 771–782.

[MMSV01] M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani, *A greedy facility location algorithm analyzed using dual fitting*, in Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, Lecture Notes in Comput. Sci. 2129, M. X. Goemans, K. Jansen, J. D. P. Rolim, and L. Trevisan, eds., Springer, Berlin, 2001, pp. 127–137.

[MYZ02] M. Mahdian, Y. Ye, and J. Zhang, *Improved approximation algorithms for metric facility location problems*, in Approximation Algorithms for Combinatorial Optimization, Lecture Notes in Comput. Sci. 2462, K. Jansen, S. Leonardi, and V. Vazirani, eds., Springer, Berlin, 2002, pp. 229–242.

[MF90] P. Mirchandani and R. Francis, eds., *Discrete Location Theory*, John Wiley and Sons, New York, 1990.

[MP00] R. R. Mettu and C. G. Plaxton, *The online median problem*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, Redondo Beach, CA, 2000, pp. 339–348.

[Rag88] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer programs*, J. Comput. System Sci., 37 (1988), pp. 130–143.

[RT87] P. Raghavan and C. D. Thompson, *Randomized rounding*, Combinatorica, 7 (1987), pp. 365–374.

[Spe87] J. Spencer, *Ten Lectures on the Probabilistic Method*, CBMS-NSF Regional Conf. Ser. in Appl. Math. 52, SIAM, Philadelphia, 1987.

[STA97] D. B. Shmoys, É. Tardos, and K. Aardal, *Approximation algorithms for facility location problems*, in Proceedings of the 29th ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 265–274.

[Svi97] M. I. Sviridenko, *private communication*, 1997.

[Svi98] M. I. Sviridenko, *private communication*, 1998.

[Svi02] M. Sviridenko, *An improved approximation algorithm for the metric uncapacitated facility location problem*, in Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 2337, W. Cook and A. S. Schulz, eds., Springer, Berlin, 2002, pp. 240–257.

SIAM J. COMPUT.
Vol. 33, No. 1, pp. 26–42

FASTER SUFFIX TREE CONSTRUCTION WITH MISSING SUFFIX LINKS*

RICHARD COLE† AND RAMESH HARIHARAN‡

Abstract. We consider suffix tree construction for situations with missing suffix links. Two examples of such situations are suffix trees for parameterized strings and suffix trees for two-dimensional arrays. These trees also have the property that the node degrees may be large. We add a new back-propagation component to McCreight's algorithm and also give a high probability hashing scheme for large degrees. We show that these two features enable construction of suffix trees for general situations with missing suffix links in $O(n)$ time, with high probability. This gives the first randomized linear time algorithm for constructing suffix trees for parameterized strings.

Key words. suffix tree, parameterized strings, two-dimensional suffix trees, dynamic perfect hashing

AMS subject classifications. 68W05, 68W20, 68W40

DOI. 10.1137/S0097539701424465

1. Introduction.

The *suffix tree* of a given string of length n is the compacted trie of all its suffixes. This tree has size $O(n)$ and can be constructed in $O(n)$ time [12, 16, 15]. Suffix trees have several applications (see [8]). One of the main applications of suffix trees is to preprocess a text in linear time so as to answer pattern occurrence queries in time proportional to the length of the query and independent of the length of the preprocessed text. The preprocessing involves building the suffix tree for the text. Next, given a query pattern, the unique path down the suffix tree traced by this pattern is determined; each leaf of the tree which lies further down from this path corresponds to an occurrence of the pattern.

Parameterized suffix trees. Baker [1] generalized the definition of suffix trees to *parameterized* strings, i.e., strings having variable characters or *parameters* in addition to the usual fixed *symbols*. The set of parameters and the set of symbols are disjoint. Two parameterized strings are said to *match* each other if the parameters in one can be *consistently* replaced with the parameters in the other to make the two strings identical. Here, consistency demands that all occurrences of a particular parameter are replaced by the same parameter and distinct parameters are replaced by distinct parameters. Baker [1] gave a definition of suffix trees for parameterized text strings t so as to facilitate answering pattern occurrence queries in time independent of the text length $|t|$.

Two-dimensional suffix trees. Giancarlo [7] generalized suffix trees to two-dimensional (2D) texts t in order to answer pattern occurrence queries (i.e., find all occurrences of a given square array p in the square text t) in time independent of $|t|$.

*Received by the editors February 19, 2001; accepted for publication (in revised form) June 5, 2003; published electronically November 14, 2003. This work was supported in part by NSF grants CCR-9800085 and CCR-0105678. A preliminary version of this paper appeared in Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, (STOC'97), ©ACM, 1997. http://doi.acm. org/10.1145/335305.335352.

http://www.siam.org/journals/sicomp/33-1/42446.html

†Courant Institute of Mathematical Sciences, New York University, 251 Mercer St., New York, NY 10012-1185 (cole@cs.nyu.edu).

‡Department of Computer Science, Indian Institute of Science, Bangalore 560012, India (ramesh@csa.iisc.ernet.in).

Suffix tree construction. There are several algorithms for constructing the suffix tree of a string drawn from a constant-sized alphabet set in $O(n)$ time. These include the algorithms by McCreight [12], Weiner [16], and Ukkonen [15]. All these algorithms exploit an important property of suffix trees; namely, each node has an outgoing suffix link.

Farach [5] showed how to construct suffix trees in $O(n)$ time even when the alphabet size was not constant but some polynomial in n. This algorithm differs from the others above in that it is not sweep-based and seems to be less critically dependent on the existence of outgoing suffix links. However, it requires renaming pairs of adjacent characters to get a string of size half that of the original string. The suffix tree for this smaller string is built recursively; Farach shows how the suffix tree of the original string can be obtained from the suffix tree of this smaller string in $O(n)$ time.

In contrast to suffix trees for strings, suffix trees for both parameterized strings and 2D arrays lack the suffix link property; i.e., there could be nodes in the tree without an outgoing suffix link defined. In addition, the node degrees in these suffix trees need not be bounded by a constant. Due to these two problems, the best constructions known until recently for suffix trees for parameterized strings [1, 11] and 2D arrays [7] took $O(n \log n)$ time in the worst case, where n is the size of the input string/array. In each case (i.e., in [1] and [7]; [11] uses a different data structure), the problem of missing suffix links was handled by using a dynamic tree data structure [14]; this data structure is used to find the insertion site of the next suffix in $O(\log n)$ time. Further, the problem of large node degrees was handled by the standard approach of maintaining a binary search tree, which also gave a $\Theta(\log n)$ overhead.

We mention here that Baker [1] gives two algorithms for constructing suffix trees for parameterized strings, one with time complexity $O(n \log n)$, as mentioned above, and another with time complexity $O(n(|\Pi|+\log|\Sigma|))$, where Π is the set of parameters and Σ is the set of symbols. Kosaraju [11] gave a faster algorithm with time complexity $O(n(\log|\Pi| + \log|\Sigma|))$, which is $O(n \log n)$ when $|\Pi| = \Theta(n)$.

Recently, Kim and Park [10] used the paradigm of Farach [5] to give an $O(n)$ time algorithm for 2D suffix tree construction (for polynomially bounded alphabet size). However, it is not clear how to apply this paradigm to the case of parameterized strings. In particular, it is not clear how the renaming of pairs of adjacent characters mentioned above can be accomplished in such a way that the suffix tree of the given string can be obtained from the suffix tree of the renamed string in $O(n)$ time.

Our contribution. We present two new tools in this paper.

(i) The first tool is aimed at tackling the problem of missing suffix links. We augment McCreight's algorithm with a new feature which copies nodes backwards (imagine suffix links as going forwards), thus adding additional nodes and suffix links to the suffix tree. Using a nontrivial accounting procedure, we show that this back-propagation adds only $O(n)$ extra nodes and accomplishes the construction of the suffix tree in $O(n)$ time even with missing suffix links. The back-propagation is similar to fractional cascading, as used in many pointer-based data structures of bounded degree (when viewed as graphs); the difficulty here is that the degrees are potentially unbounded, which appears to necessitate quite a different analysis.

(ii) The time analysis in (i) assumes that given a node x and a character a, the unique edge from x to a child of x starting with the character a is com-

putable in $O(1)$ time. To enable this for high degree nodes x, we give an extension of the dynamic version of the Fredman–Komlos–Szemerédi (FKS) perfect hashing scheme [6] which supports insertion of n items from a polynomial sized range in amortized constant time and linear space, with close to inverse exponential, i.e., $\frac{O(\log n)}{2^{\Theta(n^{1-\epsilon}/\log n)}}$, failure probability. This is in contrast to the previous expected time result of Dietzfelbinger et al. [3] and the previous result of Dietzfelbinger and Meyer auf der Heide [4], which achieves inverse polynomial failure probability. Searching for an item requires worst-case constant time. In fact, the items being in a polynomial sized range is not necessary for our hashing scheme; it suffices if they can be hashed into a polynomial sized range in linear time.

The above two tools provide a unified framework from which randomized $O(n)$ time algorithms for constructing suffix trees for regular strings, parameterized strings, and 2D arrays are easily derived. These algorithms work with high probability (for 2D arrays, the failure probability is only inverse polynomial and not close to inverse exponential as in the case of regular and parameterized strings; this higher failure probability arises in the process of casting the 2D array problem in the above unified framework). This is the first $O(n)$ time randomized algorithm for parameterized suffix tree construction; the previous best algorithms [1, 11] took $O(n \log n)$ deterministic time. The suffix trees we construct also have the property that the unique path in the tree corresponding to a given pattern string p can be found in $O(p)$ time, regardless of the degrees of the nodes.

2. The general setting. Before describing our algorithm, we describe the general setting for which our algorithm works. We need the following definitions.

Compacted trie. A compacted trie is a tree data structure defined on a collection of strings. This tree has one leaf per string in this collection, and each internal node has at least two children. Therefore, the number of nodes is linear in the number of strings in the given collection. Each edge of the tree is associated with (or labeled with) some substring of one of the strings in the given collection. The key property is that for every pair of leaves, the string formed by concatenating the edge labels on the path from the root to the least common ancestor of these two leaves is the longest common prefix of the strings associated with the two leaves.

In this paper, we are interested in compacted tries for certain kinds of string collections.

Quasi-suffix collections. An ordered collection of strings $s_1, s_2, \ldots, s_n$ is called a quasi-suffix collection if and only if the following conditions hold. Let $|s|$ denote the length of string s.

1. $|s_1| = n$ and $|s_i| = |s_{i-1}| - 1$. Therefore, $|s_n| = 1$.
2. No s_i is a prefix of another s_j.
3. Suppose strings s_i and s_j have a common prefix of length $l > 0$. Then s_{i+1} and s_{j+1} have a common prefix of length at least $l - 1$.

We will assume that all the strings are drawn from an alphabet of size polynomial in n.

Character oracles. Note that the total length of the strings in a quasi-suffix collection of n strings is $O(n^2)$, while our aim is to achieve $O(n)$ time construction for the compacted trie. Therefore, we cannot afford to read the collections explicitly. Instead, we will assume an oracle which supplies the ith character of the jth string of the collection on demand in $O(1)$ time.

Multiple quasi-suffix collections. Consider several distinct quasi-suffix collections

having l strings in all. These quasi-suffix collections constitute a multiple quasi-suffix collection if conditions 2 and 3 above hold for any pair of strings s_i, s_j over all the collections (in other words, these conditions hold for pairs within each collection and for pairs drawn from distinct collections as well).

Our main result will be the following.

THEOREM 1. *Let ϵ be any positive constant. The compacted trie of a quasi-suffix collection of n strings can be constructed in $O(n)$ time and space with failure probability at most $\frac{O(\log n)}{2^{\Theta(n^{1-\epsilon}/\log n)}}$, given the above character oracle. Further, the compacted trie of a multiple quasi-suffix collection comprising l strings in all can be constructed in $O(l)$ time and space with failure probability at most $\frac{O(\log l)}{2^{\Theta(l^{1-\epsilon}/\log l)}}$,*

2.1. Examples of quasi-suffix collections. The significance of the above theorem comes from the following examples of quasi-suffix collections. The simplest example is the collection of all suffixes of a string s with a special end-of-string symbol. This is a quasi-suffix collection but with a stronger property; namely, condition 3 in the definition of quasi-suffix collections is satisfied with equality. The compacted trie of these suffixes is the well-known suffix tree of the string s. Next, we give two more significant examples for which equality need not hold in condition 3.

2.1.1. Suffix trees for parameterized strings. Recall from the introduction that a parameterized string s has *parameters* and *symbols*. The alphabet from which parameters are derived is disjoint from the alphabet from which symbols are derived. Further, both alphabet sizes are polynomial in n, the length of s. As is standard, assume that s ends in a symbol \$ which does not occur elsewhere in s. From s, Baker [1] defined the following collection of strings.

Each suffix s' of s is mapped to a string $num(s')$ with parameters replaced by numbers and symbols retained as such (assume that symbols are not numbers). The replacement of parameters is done as follows. The first occurrence of each parameter in s' gets value 0 in $num(s')$. Subsequent occurrences of a parameter get values equal to the distance from the previous occurrence of the same parameter. Consider the collection of strings $\{num(s')|s' \text{ suffix of } s\}$ in decreasing length order. Baker [1] defined the suffix tree of parameterized string s to be the compacted trie of this collection. That this collection of strings is indeed a quasi-suffix collection can be seen as follows.

Condition 1 clearly holds, and condition 2 follows from the occurrence of the special symbol \$ at the end of s. Condition 3 is shown to hold next. Note that if s'_i and s'_{i+1} are two consecutive suffixes of s, then $num(s'_{i+1})$ can be obtained from $num(s'_i)$ as follows: for each well-defined index $k > 0$, set $num(s'_{i+1})[k]$ to $num(s'_i)[k+1]$ if $num(s'_i)[k+1] \neq k$, and set $num(s'_{i+1})[k]$ to 0 otherwise. Next, consider two suffixes s'_i and s'_j of s. From the above observation, it follows that if $num(s'_i)$ and $num(s'_j)$ have a common prefix of length $k+1$, then $num(s'_{i+1})$ and $num(s'_{j+1})$ have a common prefix of length k. Further, if $num(s'_i)$ and $num(s'_j)$ differ at location $k+1$, then $num(s'_{i+1})$ and $num(s'_{j+1})$ could be identical at location k if one of $num(s'_i)[k+1], num(s'_j)[k+1]$ equals k and the other equals 0. Condition 3 is now easily seen to hold.

The character oracle for the above quasi-suffix collection is easily implemented in $O(1)$ time after the following precomputation: for each occurrence of a parameter in s, determine the previous occurrence, if any, of this parameter in s. This precomputation is easily done in $O(n)$ time.

2.1.2. Suffix trees for 2D arrays. Consider a 2D array s having size $m \times n$, $m \geq n$ and characters drawn from some polynomial range. For each square subarray s' of s which is *maximal* (i.e., touches either the right boundary or the bottom boundary or both boundaries of s), Giancarlo [7] defined a string $num(s')$ as follows.

Defining $num(s')$. Partition s' into L's as in [7] (an L is formed by taking a prefix of a row and a prefix of a column, with the common point being at the bottom-right; both prefixes have equal lengths; the resulting shape is actually the image of the character L reflected about a vertical axis). $num(s')$ will be a sequence of numbers, with one number for each such L; these numbers are arranged in increasing order of L sizes. The number for a particular L is obtained by reading this L as a string and then mapping strings to integers in such a way that distinct strings map to distinct integers (by using, for example, the Karp–Rabin fingerprinting scheme [9], which ensures this property with inverse polynomial failure probability). Finally, a special end-of-string symbol \$ is appended to $num(s')$, as was done for parameterized strings.

The quasi-suffix collections. Consider a particular top-left to bottom-right diagonal and consider all maximal square subarrays of s with top-left point on this diagonal. The $num()$ strings corresponding to these subarrays are easily seen to form a quasi-suffix collection. Thus each top-left to bottom-right diagonal gives a quasi-suffix collection of strings. Since there are $m+n-1$ diagonals, we have $m+n-1 = O(m)$ distinct quasi-suffix collections in all. It is easy to check that these $m+n-1$ quasi-suffix collections together constitute a multiple quasi-suffix collection (we will use distinct end-of-string symbols for each diagonal to satisfy condition 2 for pairs of strings drawn from distinct collections). Note that the number of strings in each collection is at most n. Giancarlo [7] defined the common compacted trie of these $m+n-1$ collections to be the suffix tree of s.

The character oracle. A character oracle which works with inverse polynomial failure probability in $O(1)$ time after $O(mn)$ preprocessing is easy to implement using the Karp–Rabin fingerprinting scheme. The preprocessing involves computing prefix sums for each row and column.

2.2. Proving Theorem 1. The rest of the paper is devoted to proving Theorem 1. First, we will describe how to construct the compacted trie of a *single* quasi-suffix collection of n strings in $O(n)$ time with high probability. This algorithm can easily be extended to multiple quasi-suffix collections (such as those resulting from 2D arrays). This extension is sketched briefly in section 6.

Our algorithm for a single quasi-suffix collection will have two components. The first component is a modification of McCreight's algorithm and is described in section 4 and section 5. In these sections, we will assume that the unique child of any given node with edge label beginning with a given character can be determined in $O(1)$ time. The second component, i.e., a dynamic perfect hashing scheme described below, will handle this problem.

Note that in all the above examples of quasi-suffix collections, the alphabet size is a polynomial in n (while a radix sort followed by relabeling could reduce this to size at most n, the difficulty would be to subsequently process searches in the suffix tree, as the search string would be written using the "old" alphabet). Thus to access the unique edge with a particular starting character from a node, we need to perfectly hash $O(n)$ pairs, where the first entry in the pair is a node number and the second entry is a character from the alphabet. Each such pair can be treated as a number from a range polynomial in n. In section 7, we give a dynamic hashing scheme which will perfectly hash items from a polynomial in n range with close to inverse exponential

failure probability.

Before giving our algorithms, we need an outline of McCreight's algorithm for constructing the suffix tree of a string.

3. McCreight's algorithm. The use of *suffix links* is crucial to this algorithm. Suffix links are defined as follows.

Definitions. For a node x, let $str(x)$ denote the substring associated with the path from the root of the tree to x. A suffix link points from a node x to a node y such that $str(y)$ is just $str(x)$ with the first character removed. Let $link(x)$ denote this node y. Let $par(x)$ denote the parent of x. For a string u, define $node(u)$ to be that node x, if any, for which $str(x) = u$.

Since condition 3 in the definition of quasi-suffix collections is satisfied with equality for the collection of suffixes of a string, suffix links are defined for each node x in the suffix tree; i.e., for each node x, a node $y = link(x)$ with the above description exists.

McCreight's construction inserts suffixes into the suffix tree one by one in order of decreasing length. For each suffix i, one new leaf and possibly one new internal node are inserted. The algorithm for inserting suffix $i+1$, given that suffix i inserted leaf y as a child of an existing or new internal node x, is as follows.

The search for the insertion site of suffix $i+1$ begins from $link(par(x))$. It involves two stages: a *rescanning* stage and, possibly, a *scanning* stage.

In the rescanning stage, the tree is *rescanned* downwards from $link(par(x))$ until the right position for $link(x)$ is found. Rescanning requires determining that path down the tree from $link(par(x))$ whose edge labels form the same substring as the label on the edge between $par(x)$ and x. Such a path is guaranteed to exist by condition 3 in the definition of quasi-suffix collections. By virtue of this guarantee, it suffices to examine just the first character on each edge to determine this path, as opposed to examining all the characters comprising the edge label; thus we have the term rescanning (as opposed to *scanning*, which involves examining all the characters in the labels at each edge encountered).

Next, there are two cases depending on whether or not a node is already present at the position for $link(x)$ identified above. If no node is currently present, then equality in condition 3 in the definition of quasi-suffix collections demands that a new internal node be inserted at this location and a new leaf corresponding to suffix $i+1$ be inserted as its child; there is no scanning stage in this case. On the other hand, if a node is indeed present at the above position, then the algorithm involves scanning downwards from this position. In either case, note that $link(x)$ is now well defined.

The two key facts used to show $O(n)$ time performance over all suffixes are as follows. Consider the portions of the suffix tree traversed in the scanning stages for the various suffixes (we will call them *scanned* portions). These scanned portions correspond to disjoint portions of the input string, and, therefore, they sum up to $O(n)$ in length (the length of a scanned portion is the number of characters, not nodes, encountered in the path scanned). Further, the total time taken in rescan stages between any two consecutive scanning stages is bounded by the time taken in the first of these two scanning stages.

Two problems. Two related problems arise in generalizing the above algorithm to quasi-suffix collections. The first is that $link(par(x))$ may not be defined. The second is that the lack of a node at the right position for $link(x)$ (as located in the rescanning stage) no longer requires a new node to be inserted at this location (this is due to the lack of equality in condition 3 in the definition of quasi-suffix collections);

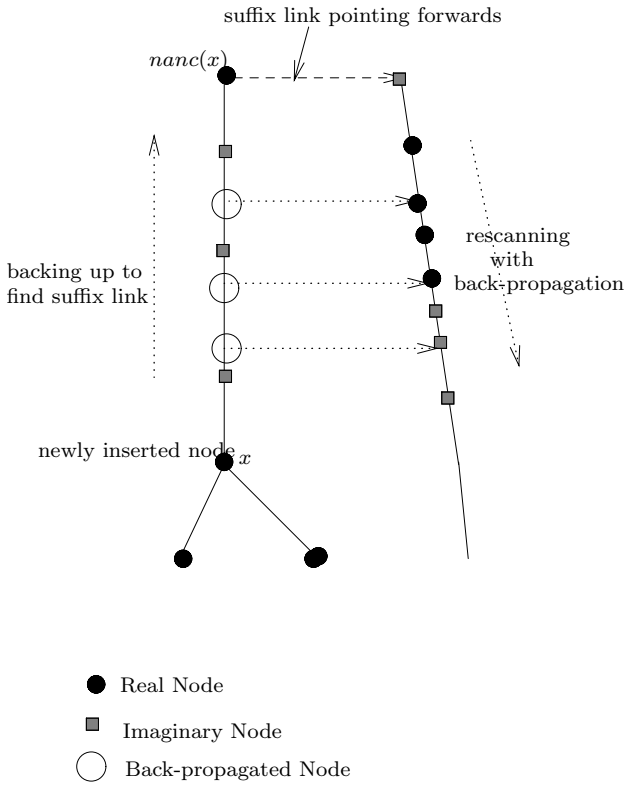


FIG. 1. *Backing up and back-propagation.*

we note that if a new node is not inserted, then a scanning stage will begin from this position.

4. Our algorithm. As in McCreight's algorithm, we will insert the strings in the given collection $s_1, \ldots, s_n$ in the compacted trie in decreasing order of length. Much of the algorithm remains the same; however, we make two key modifications. The first involves traversing the path up the tree from a newly inserted node to find an ancestor with a suffix link. The second involves copying nodes backwards while rescanning down the tree from the destination of the above suffix link. These changes affect only the rescanning algorithm; the scanning part remains unchanged. We describe these changes in detail next.

Defining suffix links. For a node x, $link(x)$ is now defined to be that node y such that if $str(x)$ is the longest common prefix of some s_i and s_j, then $str(y)$ is a common prefix of s_{i+1} and s_{j+1}; further, $|str(y)| = |str(x)| - 1$. Note that since condition 3 in the definition of quasi-suffix collections need not be satisfied with equality, $link(x)$ need not be defined for every node x. Also note that if $link(x)$ exists, then it is unique.

Backing up. Recall McCreight's algorithm above. Now, since $link(par(x))$ need not exist, we must traverse up the tree from x until a node with a suffix link is found. We call this node $nanc(x)$ (*nanc* stands for nearest ancestor). It may be that $nanc(x) = x$. Next, the tree is rescanned downwards from $link(nanc(x))$, as before, but with one modification to be described shortly. See Figure 1.

Real and imaginary nodes. Recall our description of McCreight's algorithm above. If a new scanning stage begins from the position identified for $link(x)$ in the rescanning stage, and there is no node at this position, we introduce an *imaginary node* at this position. Note that this imaginary node has only one child. Internal nodes which are not imaginary will be called *real*. Real nodes will have at least two children each; in addition, they will also have outgoing suffix links pointing, possibly, to imaginary nodes.

Note that there are just $O(n)$ real nodes and $O(n)$ imaginary nodes (at most one real internal node, one leaf, and one imaginary node are inserted per suffix). Since real nodes have at least two children each, imaginary nodes have just one child each, and the number of leaves is n, the total number of children over all real and imaginary nodes is $O(n)$. Also note that the total length of the scanned portions of the tree in McCreight's algorithm is $O(n)$, and this remains the same for our algorithm. We state these facts below for future reference.

FACT 1.

(i) *The number of real and imaginary nodes together is* $O(n)$.
(ii) *The total number of children of real and imaginary nodes together is* $O(n)$.
(iii) *The total length of the scanned portions of the tree is* $O(n)$ *(the length of a single scanned portion is the number of characters, not nodes, encountered in the path scanned).*

We need to add one more feature to McCreight's algorithm to get linear time complexity for quasi-suffix collections.

Back-propagated nodes. Other than real and imaginary nodes, our construction will involve internal nodes of a third kind, called *back-propagated nodes*. Back-propagated nodes will always have suffix links and only one child each. They are defined as follows. In the following, think of suffix links as pointing forwards (i.e., to the right; see Figure 1).

When the appropriate path starting at $link(nanc(x))$ is rescanned in order to determine the position for $link(x)$, several nodes could be encountered in the process. If more than two nodes are encountered, then alternate nodes are propagated back to the path $(nanc(x), x)$ (i.e., new nodes with suffix links pointing to the traversed nodes are set up on this path), taking care that the first and the last nodes traversed are not propagated back. The new nodes are called back-propagated nodes.

Direction of back-propagation. Note that a node could be back-propagated in several different directions; i.e., several back-propagated nodes could have their suffix links pointing to this node. Further, a back-propagated node could be propagated backwards further, forming a chain of back-propagated nodes.

Definitions. For a node x, let $prev(x)$ be a set of strings defined as follows. For each s_i in the given quasi-suffix collection having prefix $str(x)$, $prev(x)$ contains the prefix of s_{i-1} of length $|str(x)| + 1$. Note that $prev(x)$ is a set and not a multiset; therefore all strings in it are distinct. *Direction* u is said to be *valid* for node x if string u appears in $prev(x)$. Node x is said to be *back-propagated in direction* u if there exists a string u in $prev(x)$ such that $node(u)$ exists and is a back-propagated node (see Figure 2). Note that the suffix link of $node(u)$ points to x under these conditions, i.e., $link(node(u)) = x$.

The following invariant is maintained by our algorithm by virtue of the fact that only alternate nodes encountered are back-propagated and the first and last nodes encountered are not back-propagated.

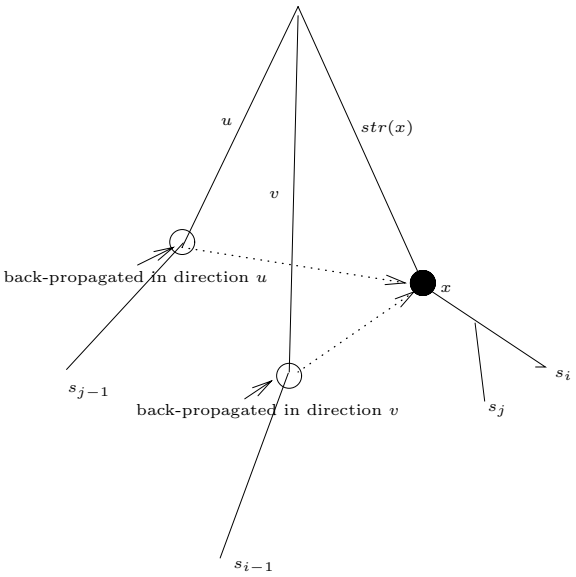


FIG. 2. *Direction of back-propagation.*

INVARIANT 1. *If a node x is back-propagated in direction u, then its parent is not back-propagated in direction u', where u' is a prefix of u. The algorithm is presented in pseudocode below.*

The algorithm is presented in pseudocode in Figure 3.

5. Time complexity. There are two aspects to the time taken to insert a particular string s_i from the given quasi-suffix collection. The first involves backing up from x to $nanc(x)$, subsequent to the insertion of x. The second involves rescanning the appropriate path down from $link(nanc(x))$ until the position for $link(x)$ is located. We account for these two aspects of the time separately.

We make a few remarks on the second aspect here. Each step taken here involves one of the following:

1. Creating a new back-propagated node.
2. Adding a suffix link to an already existing node. This happens when one seeks to back-propagate a node but the site of this back-propagation is already occupied by some other node. For this to happen, the latter node must not have a suffix link; i.e., it must be an imaginary node. A suffix link is now added to this imaginary node.
3. Creating a new real or imaginary node. This is the node $link(x)$.

Since only one real or imaginary node is added when rescanning from $link(nanc(x))$ to $link(x)$, the time taken in this rescanning is proportional to $O(1)$ plus the number of nodes back-propagated in this process plus the number of imaginary nodes for which suffix links are set up in this process. Since each imaginary node can get only one suffix link during the course of the entire algorithm, bounding the above time boils down to bounding the number of back-propagated nodes by $O(n)$.

5.1. Bounding back-propagated nodes. This will use a charging argument, where each back-propagated node will be charged to either some real/imaginary node or to some character in the string s_1. Each real/imaginary node and each character

//Insert suffix s_1
  Create a single edge (x_1, y_1) with x_1 as the root, labeled s_1.
for $i = 2$ to n **do**
  //Insert suffix s_i
    Let y_{i-1} be the leaf inserted for s_{i-1}, and let x_{i-1} be its parent.
    Find $nanc(x_{i-1})$, the nearest ancestor of x_{i-1} with a suffix link, if any; $nanc(x_{i-1})$ is the root node otherwise.
    Rescan the path starting at link $(nanc(x_{i-1}))$ until the location for link (x_{i-1}) is reached.
    If link (x_{i-1}) is a new node, split the label on the edge previously containing link (x_{i-1})'s location so that $|string(link(x_{i-1}))| = |string(x_{i-1})| - 1$.
    As the rescan proceeds, back propagate every second node encountered, starting two nodes after $link(nanc(x_{i-1}))$ and stopping two nodes before $link(x_{i-1})$.
    If node b is the back propagation of node c, split the label on the edge previously containing b's location so that $|string(b)| = |string(c)| + 1$.
    If $link(x_{i-1})$ was already present, scan from $link(x_{i-1})$ to find the location for y_i.
    Add a label for the edge to node y_i so that $string(y_i) = s_i$.
od

FIG. 3. *Algorithm for quasi-suffix tree construction.*

in s_1 will be charged $O(1)$ in the process. The $O(n)$ bound will follow from Fact 1. It may be that a node created by back-propagation subsequently becomes real or imaginary. These nodes are not counted; only nodes that are not real or imaginary when the full tree is built are counted.

Note that a back-propagation chain always starts at a real or an imaginary node. We will define a tree for each real or imaginary node x as follows.

Defining $BP - tree(x)$. All nodes in this tree other than the root x are back-propagated nodes. Those back-propagated nodes which are back-propagated from x (i.e., have suffix links pointing to x) are children of x in this tree. Trees rooted at these children are defined recursively; i.e., children of a node are those which are back-propagated from that node. The leaves of this tree are those nodes from which no further back-propagation occurs.

Consider the forest of $BP - trees(*)$ rooted at the various real/imaginary nodes that are back-propagated. Each back-propagated node appears in exactly one tree in this forest.

Decomposing $BP - tree(x)$ *into paths.* We partition the nodes of this tree into paths. The first path is the minimal path starting from the root x and ending on a node y with the following property: either there exists a valid direction u such that y has not been back-propagated in this direction or there is no valid direction for y. Clearly, such a node y must exist. But for the termination restriction, the path starting at the root is chosen arbitrarily. Once nodes in this path are removed, the subtrees hanging off this path are decomposed recursively.

Clearly, each back-propagated node will belong to exactly one of the various paths formed above. Think of each path as going backwards from its start node.

Accounting for long paths. We show that the sum of the lengths of all the paths obtained above is proportional to the number of such paths plus $O(n)$. It will then

suffice to bound the number of such paths.

Consider any path obtained above. Let x be any node on this path other than its start node. $link(x)$ is the node from which x was back-propagated, say in direction u. Note that $link(x)$ will precede x in the path being considered (as paths go backwards).

By Invariant 1, the parent $par(link(x))$ of $link(x)$ in the compacted trie has not been back-propagated in the direction u', where u' is the prefix of u such that $|u'|$ equals $|str(par(link(x)))| + 1$; u', of course, is a valid direction for $par(link(x))$ (because u is valid for $link(x)$ itself). It follows that either $par(link(x))$ is a real/imaginary node or $par(link(x))$ is a back-propagated node and the last node in its path (for if there is a direction in which a node is not back-propagated, then by construction that node is the last node on its path). In either case, we charge $par(link(x))$ for x.

Clearly, in this process each real/imaginary node and each back-propagated node which is the last node in its respective path will be charged an amount bounded by the number of its children. From Fact 1(ii), this charge sums to $O(n)$ for real/imaginary nodes. Note that back-propagated nodes have only one child each. Thus, it now suffices to bound the total number of paths.

Bounding the total number of paths. We will extend the above paths backwards to form a collection of *extended paths*, as below.

Consider any one path, and let x be the last node on this path. The extension to this path is performed as follows. Start at x and follow that direction backwards along which x was not back-propagated (there is at least one such direction, unless there are no valid directions for x). Next, repeatedly follow any arbitrarily chosen valid direction backwards. This extension need not always encounter a node (in fact we will stop when we hit a node); it is allowed to cut through edges.[1] So if a particular step of this extension leads to the middle of an edge e, take an arbitrary valid direction back from that point on e. Continue this extension until either a node is reached or there is no valid direction along which to continue.

Thus an extended path consists of an initial prefix of nodes (i.e., the path itself), followed by a walk which cuts through edges, and possibly terminates on a node. Again, note that we think of a path as going backwards. We have the following claims.

LEMMA 5.1. *Two distinct extended paths cannot intersect (i.e., they cannot cut through the same point on some edge or have a node in common), except that the last node of one can be the first node of the other.*

Proof. Since forward directions are always unique, two extended paths can intersect otherwise only if the start node of one path is contained in the other path and is not the last node on that path. This is a contradiction since all the unextended paths begin at nodes, the unextended paths are node disjoint, and the extension of a path terminates as soon as a node is reached. □

LEMMA 5.2. *If an extended path terminates by reaching a node y (and not by running out of valid directions), then y cannot be a back-propagated node.*

Proof. Let x be the last node of the path whose extension is under consideration. Suppose y is a back-propagated node. As forward links are unique, clearly x must have been back-propagated in the direction implied by y. But we started the extension of this path by choosing a direction along which x was not back-propagated, a contradiction. □

[1] We have defined valid directions only for nodes in the compacted trie. However, this definition can be extended for points in the middle of an edge in the obvious way, i.e., by imagining a node to be present at that point.

LEMMA 5.3. *The total number of paths is $O(n)$, and hence the total number of back-propagated nodes is $O(n)$.*

Proof. Consider a particular extended path. If it ends at a node without running out of valid directions, this node must be real/imaginary by Lemma 5.2; the current path is then charged to this node. By Lemma 5.1, each real/imaginary node is just charged once.

On the other hand, if this extended path ends because all further valid directions backwards are exhausted, then the substring associated with the termination point is a prefix of s_1. Further, by Lemma 5.1, different extended paths which end in this way are associated with distinct prefixes of s_1. Thus the number of paths is $O(n)$.

The lemma follows from the argument given earlier that the number of back-propagated nodes is proportional to the number of paths plus $O(n)$. ∎

5.2. Backing-up time. It remains to account for the time taken to determine $nanc(x)$ after the insertion of a leaf as a child of x. Note that all such nodes x for which $nanc(x)$ will be determined are real nodes (because x has at least two children).

This computation requires traversing upwards from x until the nearest node with a suffix link is found. All nodes encountered on the way must be imaginary (real and back-propagated nodes have suffix links), and we need to account for the time taken to traverse these nodes.

The key claim is the following. Note that an imaginary node y signals the beginning of a new scanning phase in McCreight's algorithm, in which the tree is scanned downwards starting at y, until a new leaf is inserted as a child of a new or existing internal node z.

LEMMA 5.4. *The total number of times imaginary node y can be encountered while determining $nanc(*)$ over the entire algorithm is at most $|str(z)| - |str(y)|$.*

Proof. Note that z is a real node after the above scanning phase starting at y finishes. y could be encountered once while setting up $link(z)$. Subsequently, since $link(z)$ is in place, y will be encountered only when finding $nanc(z')$, where z' is real and on the path from y to z. There can be at most $|str(z)| - |str(y)|$ such distinct real nodes z'. ∎

COROLLARY 5.5. *The total time taken in traversing imaginary nodes while determining $nanc(*)$ is $O(n)$.*

Proof. $|str(z)| - |str(y)|$ equals the number of characters scanned in the scanning phase following the insertion of imaginary node y. By Fact 1(iii), summed over all y, this is $O(n)$ characters. But by Lemma 5.4, summed over all y, this is also the number of imaginary nodes encountered while determining $nanc(*)$. ∎

Theorem 1 now follows for quasi-suffix collections, assuming that the correct child of a particular node can be found in $O(1)$ time. The extension to quasi-suffix collections is sketched next.

6. Algorithm for multiple quasi-suffix collections. We sketch how to extend the above algorithm to a multiple quasi-suffix collection having l strings in all. The time taken will be $O(l)$.

Suffix links and back-propagation directions need to be redefined appropriately as follows. Let s_i^k denote the ith string in the kth quasi-suffix collection under consideration (assume an arbitrary ordering on the various quasi-suffix collections).

Suffix links. For a node x, $link(x)$ is now defined to be that node y such that if $str(x)$ is the longest common prefix of some s_i^k and s_j^l, then $str(y)$ is a common prefix of s_{i+1}^k and s_{j+1}^l; further, $|str(y)| = |str(x)| - 1$. Note that since condition 3 in the

definition of quasi-suffix collections need not be satisfied with equality, $link(x)$ need not be defined for every node x. Also note that if $link(x)$ exists, then it is unique; this follows because if $str(x)$ is a prefix of s_i^k and of $s_j^{k'}$, then s_{i+1}^k and $s_{j+1}^{k'}$ agree in the first $|str(x)| - 1$ characters.

Back-propagation directions. For a node x, let $prev(x)$ be a set of strings defined as follows. For each s_i^k having prefix $str(x)$, $prev(x)$ contains the prefix of s_{i-1}^k of length $|str(x)| + 1$. Note that $prev(x)$ is a set and not a multiset; therefore all strings in it are distinct. *Direction* u is said to be *valid* for node x if string u appears in $prev(x)$.

The algorithm. The algorithm inserts each collection in turn into the current compacted trie. The first string of each quasi-suffix collection starts a new scanning stage beginning at the root of the compacted trie. The subsequent strings in the collection are inserted as in the previous algorithm. Note that the size of the compacted trie will now be $\Theta(l)$. Fact 1 continues to hold with $O(n)$ replaced by $O(l)$. The analysis is as before with the following two changes. All $O(n)$ terms are replaced by $O(l)$. Further, in Lemma 5.3, if an extended path ends because all further valid directions backwards are exhausted, then the substring associated with the termination point is a prefix of the first string in one of the several quasi-suffix collections being considered.

7. The hashing scheme. Recall from section 2.2 that we need to perfectly hash $O(n)$ pairs, where the first entry in each pair is a node number and the second entry is a character from the alphabet. Each such pair can be treated as a number from a range polynomial in n. We give a dynamic hashing scheme which will perfectly hash an item from a polynomial in n range in amortized $O(1)$ time, with close to inverse exponential failure probability. The time taken to access a particular item will be $O(1)$, and the total space is $O(n)$.

Fredman, Komlos, and Szemerédi [6] showed how n items from the range $[0 \ldots poly(n)]$ can be hashed into the range $[0 \ldots s]$ without any collisions, where $s = \Theta(n)$. Their algorithm takes $O(n)$ time and space and works by choosing randomly from a family of almost-universal hash functions (assuming constant time arithmetic on $O(\log n)$ bits). It ensures no collisions with probability at least $1/2$.

This was generalized by Dietzfelbinger et al. [3] to the dynamic setting. The expected amortized insertion/deletion time for their algorithm is $O(1)$; searching takes $O(1)$ worst-case time. Subsequently, Dietzfelbinger and Meyer auf der Heide [4] achieved $O(1)$ worst-case insertion/deletion/search time with inverse polynomial failure probability. We achieve close to inverse exponential failure probability but with $O(1)$ amortized insertion/deletion times and $O(1)$ worst-case search time. This is done by modifying the FKS perfect hashing scheme to make it work with high probability, first in the static setting and then in the dynamic setting.

First, we present the static algorithm. The key idea is to create several perfect hashing subproblems and to apply the FKS scheme on each independently to obtain a high success probability.

7.1. The static hashing scheme. The following steps are performed. Let ϵ be any positive constant. The time and space taken by our data structure will be linear but with a $\frac{1}{\epsilon}$ constant factor. The failure probability will decrease as ϵ gets closer to 0.

Step 1. Start with an imaginary array A of size n^c, where the n items to be hashed come from the range $1 \ldots n^c$. Each item indexes into a unique element in this array. Next, repeatedly partition this array as in Step 2.

Step 2. Construct a *partition tree* as described below. Each node in this tree will have a subarray of A associated with it. The depth of this tree will be a constant, and the number of nodes will be $O(n)$. The root of this tree is A itself. It has n^ϵ children, each associated with a distinct subarray of A of size $n^{c-\epsilon}$ obtained by partitioning A into n^ϵ disjoint pieces. Each subarray with more than n^ϵ items is recursively partitioned; the remaining subarrays become leaves. Each leaf has at most n^ϵ items. Clearly, the number of levels in this tree is $O(\frac{c}{\epsilon}) = O(1)$, and the total size in $O(n)$. The total time taken to set up the tree is easily seen to be $O(n)$.

Step 3. Next, we consider each leaf of the above tree in turn and the items in the subarray associated with this leaf. We perfect-hash these items using the FKS perfect hashing scheme. Since this scheme succeeds only with probability 1/2, several trials may be required before these items are perfectly hashed. We show that with high probability, the total time taken in this process over all leaves is $O(n)$.

7.2. Time complexity. We need to bound the time taken to perform several FKS perfect hashings, where the total sizes of all subproblems is n, each subproblem has size at most n^ϵ, and a subproblem is performed successfully in linear time with probability 1/2.

Size categories. Divide the leaves into $O(\log n)$ categories quadrupling by size (i.e., the number of items associated with the leaf). Consider just leaves in any one size category, namely, the category in which leaf sizes are in the range $\frac{n^\epsilon}{4^{i+1}} \cdots \frac{n^\epsilon}{4^i}$, $i \geq 0$. We will show that the time taken for this category is proportional to the sum of the sizes of leaves in this category plus $O(\frac{n}{2^i})$, with failure probability $\frac{O(\log n)}{2^{\Theta(\frac{2^i n^{1-\epsilon}}{\log n})}}$. It follows that the total time taken over all categories is $O(n)$, with failure probability $\frac{O(\log n)}{2^{\Theta(\frac{n^{1-\epsilon}}{\log n})}}$.

A leaf is said to *succeed* when the items in it are perfectly hashed. A *round* refers to one trial for each of the relevant leaves. The trials for the various leaves can be imagined to have proceeded in rounds, with leaves succeeding in one round dropping out of the subsequent rounds. We organize the rounds into groups.

Grouping rounds. The 0th group comprises rounds performed before the number of unsuccessful leaves in this size category drops below $\frac{n^{1-\epsilon}2^i}{\log n}$. For $j \geq 1$, the jth group comprises rounds performed after the number of unsuccessful leaves in this size category drops below $\frac{n^{1-\epsilon}2^i}{2^{j-1}\log n}$ but before this number drops below $\frac{n^{1-\epsilon}2^i}{2^j \log n}$.

We show that group 0 has $O(i + \log\log n)$ rounds and that each group $j \geq 1$ has $O(2^j)$ rounds, with failure probability $\frac{O(\log n)}{2^{\Theta(\frac{n^{1-\epsilon}2^i}{\log n})}}$ (over all groups). Further, we show that with the same failure probability, every two consecutive rounds in group 0 reduce the number of unsuccessful leaves by half. The total time taken for rounds in group 0 is then proportional to the sum of leaf sizes in this category. The time taken for rounds in the other groups is

$$O\left(\sum_{j=1}^{\Theta(\log n)} [2^j \frac{n^{1-\epsilon}2^i}{2^{j-1}\log n}\frac{n^\epsilon}{4^i}]\right) = O\left(\frac{n}{2^i}\right),$$

as required.

The key property. To show the above claims on the number of rounds in each group, we will need the following property, obtained using the Chernoff bound [2]. If there are $\#u$ unsuccessful leaves at some instant of time, then half these leaves succeed in the next $2k$ rounds, with failure probability $\frac{1}{2^{\Theta(\#uk)}}$.

Group 0. First, consider group 0. If the number of unsuccessful leaves at some instant is at least $\frac{n^{1-\epsilon}2^i}{\log n}$, then two rounds will halve the number of unsuccessful leaves, with failure probability at most $\frac{1}{2^{\Theta(\frac{n^{1-\epsilon}2^i}{\log n})}}$ (apply the above property with $k = 1$ and $\#u \geq \frac{n^{1-\epsilon}2^i}{\log n}$). Note that the number of leaves in the size category being considered is at most $\frac{n}{n^\epsilon/4^{i+1}} = n^{1-\epsilon}4^{i+1}$ to begin with. It follows that group 0 has $2(i+2+\log\log n)$ rounds, and halving occurs in each pair of consecutive rounds, with failure probability at most $\frac{(i+2+\log\log n)}{2^{\Theta(\frac{n^{1-\epsilon}2^i}{\log n})}} = \frac{O(\log n)}{2^{\Theta(\frac{n^{1-\epsilon}2^i}{\log n})}}$.

Other groups. Next, consider group j, $j \geq 1$. Applying the above property with $k = 2^j$ and $\#u \geq \frac{n^{1-\epsilon}2^i}{2^j \log n}$, we get that group j has $2\cdot 2^j$ rounds, with failure probability $\frac{1}{2^{\Theta(\frac{n^{1-\epsilon}2^i 2^j}{2^j \log n})}} = \frac{1}{2^{\Theta(\frac{n^{1-\epsilon}2^i}{\log n})}}$. Finally, adding up the failure probability over all $O(\log n)$ groups gives $\frac{O(\log n)}{2^{\Theta(\frac{n^{1-\epsilon}2^i}{\log n})}}$, as required.

The total time and space taken above is thus $O(n)$, with high probability. Searching for an element requires following the unique path down the partition tree to reach the relevant perfect-hash table. These operations are easily seen to take $O(1)$ worst-case time.

Comment. This analysis can also be applied to the second stage of the standard FKS scheme, assuming the first stage has succeeded (i.e., the initial hash has partitioned the items so that the expected number of pairwise collisions is $O(n)$, and so every bucket holds $O(n^{1/2})$ items). We then conclude that the second stage fails with close to exponentially small probability.

This might lead one to consider a high probability 3-stage FKS-like scheme. The first stage will be the standard FKS first stage, but it will be decreed to succeed if the number of pairwise collisions is at most $n^{3/2}$. This happens with probability $1 - O(1/n^{1/2})$ by Markov's inequality. This step can be repeated up to $2d$ times to obtain a failure probability of $O(1/n^d)$. The sets resulting from the first stage are then hashed using a standard 2-stage FKS scheme, but as each of these sets has size $O(n^{3/4})$, by an analysis similar to the one of this section one obtains a close to exponentially small failure probability. Thus the overall failure probability is $O(1/n^d)$. Note that as the first stage is repeated only if necessary, this appears to entail fewer arithmetic steps than using a $2d$-independent hash function.

7.3. The dynamic hashing scheme. The dynamic version of the above static scheme maintains the partition tree described in Step 2 above at each instant (with the same parameters; i.e., A has size n^c and the branching factor is n^ϵ; here n is the total number of items which will ever be inserted).

Initially, the partition tree will have just an empty root node. This tree will build up as insertions are made. The size of the partition tree at any instant will be proportional to the number of items in it. Further, at each instant, the perfect-hash structure at any leaf will have an associated *capacity*. This capacity will be at least the number of items at that leaf but no more than twice this quantity. It follows that the total space required at any instant will be proportional to the number of items present.

The algorithm for an insertion is described next. Note that our compacted tree application involves only insertions and no deletions.

Insertions. On an insertion x, the path down this partition tree to the appropriate leaf v is traced in $O(1)$ time. Subsequently, there are two cases depending upon how many items are already present in this leaf v.

First, suppose v has more than n^ϵ items, including x. Then the subarray associated with v is subdivided as in Step 2 of the static algorithm, and the subtree rooted at v is developed. Each leaf in this tree will have at most n^ϵ elements in it. The elements in each of these leaves are then perfect-hashed.

Next, suppose v has at most n^ϵ items, including x. Then the items already in v would have been perfect-hashed; further, this perfect-hash structure will have a certain capacity. If this capacity is equaled by the insertion of x, then all the items in v (including x) are rehashed into a perfect-hash structure of twice the capacity. Otherwise, if this capacity is not equaled, then v is perfect-hashed. If there is no collision, then v's insertion is complete. Otherwise, if there is a collision, then all the items in v along with x are perfect-hashed again.

Time analysis. We will show that the total time taken to perform n insertions is $O(n)$, with failure probability at most $\frac{O(\log n)}{2^{\Theta(n^{1-\epsilon}/\log n)}}$. To show the above, the following facts need to be noted.

1. The height of the partition tree is $O(1)$; therefore, the time spent in developing leaves into subtrees on insertion is just $O(n)$ over all n insertions.
2. The perfect-hash structure at any leaf in the partition tree begins with capacity which is twice the number of items currently in the structure. Future insertions increase this number until it equals the capacity. Until this happens, this perfect-hash structure stays in place, though it may have to be rebuilt as many times as collisions are caused by insertions. Once the number of items matches the capacity, this perfect-hash structure is abandoned, and a new perfect-hash structure with twice the capacity is put in place.
3. The total capacities of all perfect-hash structures which were ever in existence at any time during the n insertions is $O(n)$ (note that when a perfect-hash structure at a leaf is replaced by a new structure with twice the capacity, each structure is counted separately in the above sum). This follows from the doubling of capacities at a leaf and from the constant depth of the partition tree.
4. When the capacity of a perfect-hash structure at a leaf is doubled, the probability that this structure needs rebuilding before the number of items in it equals the new capacity is at most $1/2$. Further, the time taken for rebuilding a particular perfect-hash structure is proportional to its capacity.
   Note the difference from the static case, where a perfect-hash trial succeeds on the items currently present with probability $1/2$. Now, this is replaced by the fact that a perfect-hash trial succeeds with probability $1/2$ even on future insertions as long as the capacity is not equaled.

Thus, to establish the total time bound above, it suffices to bound the total time taken for rebuilding the perfect-hash structures at the various leaves. This in turn boils down to the following question: What is the total time taken to perform several FKS perfect hashings, where the total sizes of all subproblems is $\Theta(n)$, each subproblem has size at most n^ϵ, and a subproblem is performed successfully in linear time with probability $1/2$? The analysis is now identical to the static case.

We conclude with two remarks on generalizing the above scheme when the number of items is unknown and deletions need to be performed as well. Neither of these is relevant to our application of constructing suffix trees.

Unknown number of items. Suppose the number of items to be hashed is an unknown quantity m, with each item coming from the range $1 \ldots n^c$. Then we start with an initial estimate of 1 and double the estimate each time it is equaled by

insertions. Suppose the current estimate is $2e$, and the number of items inserted is e. We first hash these items into an imaginary array A of size $(2e)^c$. No collisions occur, with inverse polynomial (in e) failure probability (using families of almost-universal hash functions). We repeatedly try new hash functions until no collisions occur. Subsequently, we build the partition tree with degree $(2e)^\epsilon$. When the number of insertions equals $2e$, we double our estimate to $4e$ and rebuild the entire structure. If the total number of insertions is m, then the total time and space required is $O(m)$, with probability 1 minus an inverse polynomial in m. This failure probability can be reduced to $\frac{1}{m^{\Theta(\log m)}}$ by using a family of hash functions defined by Siegel [13], instead of a family of almost-universal hash functions.

Deletions. Deletions can be easily handled as follows. A deleted item is just marked as deleted, without causing any other change to the data structure. Whenever the number of items marked as deleted becomes a constant fraction of the number of items currently in the data structure the entire structure is rebuilt on the undeleted items. The running time remains $O(m)$ for m insertions and deletions, with the same failure probability as above. The space at any instant is proportional to the number of undeleted items.

REFERENCES

[1] B. Baker, *Parameterized pattern matching: Algorithms and applications*, J. Comput. System Sci., 52 (1996), pp. 28–42.

[2] H. Chernoff, *A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations*, Ann. Math. Statistics, 23 (1952), pp. 493–507.

[3] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan, *Dynamic perfect hashing: Upper and lower bounds*, SIAM J. Comput., 23 (1994), pp. 738–761.

[4] M. Dietzfelbinger and F. Meyer auf der Heide, *A new universal class of hash functions and dynamic hashing in real time*, in Proceedings of the 17th International Colloquium on Automata, Languages, and Programming, Warwick, England, 1990, pp. 6–19.

[5] M. Farach, *Optimal suffix tree construction with large alphabets*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 137–143.

[6] M. L. Fredman, J. Komlos, and E. Szemerédi, *Storing a sparse table with $O(1)$ worst case access time*, J. Assoc. Comput. Mach., 31 (1984), pp. 538–544.

[7] R. Giancarlo, *A generalization of the suffix tree to square matrices, with applications*, SIAM J. Comput., 24 (1995), pp. 520–562.

[8] D. Gusfield, *Algorithms on Strings, Trees and Sequences*, Cambridge University Press, Cambridge, UK, 1997.

[9] R. Karp and M. Rabin, *Efficient randomized pattern-matching algorithms*, IBM J. Res. Develop., 31 (1987), pp. 249–260.

[10] D. K. Kim and K. Park, *Linear time construction of 2-D suffix trees*, in Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, Prague, Czech Republic, 1999, pp. 463–472.

[11] S. R. Kosaraju, *Faster algorithms for the construction of parameterized suffix trees*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 631–637.

[12] E. M. McCreight, *A space economical suffix tree construction algorithm*, J. Assoc. Comput. Mach., 23 (1976), pp. 262–272.

[13] A. Siegel, *On universal classes of fast high performance hash functions, their time space trade-off, and their applications*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, Research Triangle Park, NC, 1989, pp. 20–25.

[14] D. Sleator and R. Tarjan, *A data structure for dynamic trees*, J. Comput. System Sci., 26 (1983), pp. 362–391.

[15] E. Ukkonen, *On-line construction of suffix trees*, Algorithmica, 14 (1995), pp. 249–260.

[16] P. Weiner, *Linear pattern matching algorithms*, in Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory, Iowa City, IA, 1973, pp. 1–11.

SIAM J. COMPUT.
Vol. 33, No. 1, pp. 43–68

NONDETERMINISTIC COMMUNICATION WITH A LIMITED NUMBER OF ADVICE BITS*

JURAJ HROMKOVIČ† AND GEORG SCHNITGER‡

Abstract. We present a new technique for differentiating deterministic from nondeterministic communication complexity. As a consequence we give almost tight lower bounds for the nondeterministic communication complexity with a restricted number of advice bits. In particular, for any function $t : \mathbb{N} \to \mathbb{N}$ with $t(n) \leq n/2$ we construct a family $(L_{n,t(n)} : n \in \mathbb{N})$ of languages such that

- $L_{n,t(n)} \subseteq \{0,1\}^{2n}$,
- $\mathrm{nc}(L_{n,t(n)}) = O(t(n) \cdot \log_2 \frac{n}{t(n)})$ and $\mathrm{nc}(\overline{L_{n,t(n)}}) = O\big(\frac{n}{t(n)\cdot \log_2 \frac{n}{t(n)}} + \log_2 t(n)\big)$,
- but $\mathrm{nc}_{o(t(n))}(L_{n,t(n)}) = \Omega\big(\frac{n}{\log_2 \frac{n}{t(n)}}\big)$.

Here $\mathrm{nc}_r(L)$ is the nondeterministic communication complexity of L, assuming that at most r advice bits are utilized. Thus, in contrast to probabilistic communication complexity, a small reduction in the number of advice bits results in almost maximal communication. As a special case we obtain a family $L_n \subseteq \{0,1\}^{2n}$ of languages with

$$\mathrm{nc}_{o(\sqrt{n}/\log_2 n)}(L_n) = \Omega\left(\frac{n}{\log_2 n}\right),$$
$$\mathrm{nc}(L_n) + \mathrm{nc}(\overline{L_n}) = O(\sqrt{n}),$$

and hence nondeterministic communication with slightly restricted access to advice bits is almost quadratically weaker than nondeterminism that always gives correct answers (from the set {yes, no, ?}). As a consequence we obtain an almost optimal separation between Monte-Carlo communication and "correct" nondeterminism and answer a question of Beame and Lawry.

Key words. communication complexity, nondeterminism

AMS subject classifications. 03D15, 68Q25, 68Q15

DOI. 10.1137/S0097539702414622

1. Introduction.

Communication complexity for two-party protocols [1, 21] is one of the most thoroughly investigated complexity measures (see, for instance, the textbooks [9, 15]). Communication complexity is closely related to fundamental complexity measures of several parallel and sequential computational models, i.e., Boolean circuits [19], VLSI circuits [16], Turing machines [14], branching programs and pseudorandomness [4], and combinatorial optimization [20].

An additional reason to investigate communication complexity is to determine exact relationships between the computation modes determinism, nondeterminism, and probabilism [2, 3, 11, 17]. In this paper we consider the number of advice bits for nondeterministic two-party communication as a computational resource. The related problem, namely, restricting the number of random bits for probabilistic two-party communication, has been considered in [6, 7, 18]. In [18] it was shown that a logarithmic number of random bits suffices for optimal probabilistic (i.e., Las Vegas, Monte-Carlo and bounded-error) protocols. We show that the situation for nondeterministic communication is completely different.

*Received by the editors September 18, 2002; accepted for publication (in revised form) July 21, 2003; published electronically November 14, 2003. A preliminary version of this work originally appeared as [10]. The authors were partially supported by DFG grants HR 14/6-1 and SCHN 503/2-1.

http://www.siam.org/journals/sicomp/33-1/41462.html

†Lehrstuhl für Informatik I, RWTH Aachen, 52056 Aachen, Germany (jh@cs.rwth-aachen.de).

‡Institut für Informatik, Johann Wolfgang Goethe-Universität Frankfurt, 60054 Frankfurt am Main, Germany (georg@thi.informatik.uni-frankfurt.de).

Let $L \subseteq \{0,1\}^{2n}$ be a given language. We assume the fixed-partition model of communication complexity; i.e., processor A (resp., B) receives an input $x \in \{0,1\}^n$ (resp., $y \in \{0,1\}^n$), and both processors try to determine whether $(x,y) \in L$. We assume that an advice bit is given to a processor that requests it; hence we work within the model of *private* nondeterministic communication as opposed to the public model, in which both processors know the advice bits. For a natural number $r \in N$, let

$$\mathrm{nc}_r(L)$$

be the nondeterministic communication complexity of L restricted to protocols that do not require more than r advice bits. $\mathrm{nc}(L) = \mathrm{nc}_n(L)$ denotes the conventional nondeterministic communication complexity, whereas $\mathrm{c}(L) = \mathrm{nc}_0(L)$ is the deterministic communication complexity of L. We can now state our main result.

THEOREM 1.1. *Let $t : \mathbb{N} \to \mathbb{N}$ be an arbitrary function with $t(n) \leq n/2$. Then there is a family $L_{n,t(n)} \subseteq \{0,1\}^{2n}$ of languages with*

$$\begin{aligned}
\mathrm{nc}(L_{n,t(n)}) &= O\left(t(n) \cdot \log_2 \frac{n}{t(n)}\right), \\
\mathrm{nc}(\overline{L_{n,t(n)}}) &= O\left(\frac{n}{t(n) \cdot \log_2 \frac{n}{t(n)}} + \log_2 t(n)\right), \quad \textit{or alternatively} \\
\mathrm{nc}_{\log_2 t(n)}(\overline{L_{n,t(n)}}) &= O\left(\frac{n}{t(n)} + \log_2 t(n)\right), \quad \textit{but} \\
\mathrm{nc}_{o(t(n))}(L_{n,t(n)}) &= \Omega\left(\frac{n}{\log_2 \frac{n}{t(n)}}\right).
\end{aligned}$$

Observe that $\mathrm{nc}_r(L) = \mathrm{nc}(L)$, provided $\mathrm{nc}(L) \leq r$. Thus the nondeterministic communication of $L_{n,t(n)}$ remains bounded by $s = O(t(n) \cdot \log_2 \frac{n}{t(n)})$, even if at most s advice bits are available, and Theorem 1.1 describes an almost maximal blow-up in communication, when the number of advice bits is reduced by a logarithmic factor.

Nondeterministic communication can be restricted to being one-way without increasing communication costs. Klauck [13], however, shows a round hierarchy for nondeterministic communication with slightly restricted access to advice bits.

The crucial dependence on the number of advice bits is not just limited to nondeterministic communication, but applies to *self-verifying nondeterminism* as well, where three types of answers may be given: *yes*, *no*, and *I do not know*. Whenever the answer is "yes" or "no," then the answer has to be correct, and for each input one of these answers has to be given by at least one computation.

It is not difficult to see that svnc(L), the communication complexity of self-verifying protocols for a language L, satisfies the inequality

$$\max\{\mathrm{nc}(L), \mathrm{nc}(\overline{L})\} \leq \mathrm{svnc(L)} \leq \max\{\mathrm{nc(L)}, \mathrm{nc}(\overline{\mathrm{L}})\} + 1,$$

and we get $\mathrm{c}(L) = O(\mathrm{svnc(L)}^2)$ [2]. Our next result shows that this quadratic gap also appears between self-verifying nondeterminism and nondeterminism with slightly restricted access to advice bits. If we impose the restriction $t(n) \leq \sqrt{n}/\log_2 n$ in Theorem 1.1, then the nondeterministic communication complexity of $\overline{L_{n,t(n)}}$ dominates the nondeterministic communication complexity of $L_{n,t(n)}$, and hence $\mathrm{svnc}(L_{n,t(n)}) = O(\frac{n}{t(n) \cdot \log_2 n})$. Moreover, we can reduce the consumption of advice bits for $\overline{L_{n,t(n)}}$ by choosing the alternative version, and we get the following.

COROLLARY 1.2. *Let $t : \mathbb{N} \to \mathbb{N}$ be an arbitrary function with $t(n) \leq \sqrt{n}/\log_2 n$. Then there is a family $L_{n,t(n)} \subseteq \{0,1\}^{2n}$ of languages with*

$$\begin{aligned} \operatorname{svnc}(L_{n,t(n)}) &= O\left(\frac{n}{t(n)\cdot \log_2 n}\right), \quad \textit{or alternatively} \\ \operatorname{svnc}_{t(n)\cdot\log_2 n}(L_{n,t(n)}) &= O\left(\frac{n}{t(n)}\right), \quad \textit{but} \\ \operatorname{nc}_{o(t(n))}(L_{n,t(n)}) &= \Omega\left(\frac{n}{\log_2 n}\right). \end{aligned}$$

If we set $t(n) = \sqrt{n}/\log_2 n$ and $L_n = L_{n,t(n)}$, then the almost quadratic and thus almost maximal blow-up between slightly restricted nondeterministic communication and self-verifying nondeterminism becomes apparent.

COROLLARY 1.3. *There is a family $L_n \subseteq \{0,1\}^{2n}$ of languages with*

$$\begin{aligned} \operatorname{svnc}(L_n) &= O(\sqrt{n}), \quad \textit{but} \\ \operatorname{nc}_{o(\sqrt{n}/\log_2 n)}(L_n) &= \Omega\left(\frac{n}{\log_2 n}\right). \end{aligned}$$

Thus nondeterministic and self-verifying nondeterministic protocols show a completely different behavior compared with their probabilistic counterparts, Monte-Carlo and Las Vegas protocols, which require at most $O(\log_2 n)$ random bits [18].

Beame and Lawry ([5]; see also problem 3.11 in [15]) ask whether the communication complexity of self-verifying nondeterministic protocols is an asymptotic upper bound for the Las Vegas communication complexity. We show that Corollary 1.3 implies a negative answer even for Monte-Carlo communication. Let $\operatorname{mcc}(L)$ be the Monte-Carlo communication complexity of L, where we assume error probability $\frac{1}{2}$. By [18] we have $\operatorname{mcc}(L) \geq \operatorname{nc}_{O(log_2 n)}(L) - O(\log_2 n)$, and hence Corollary 1.3 implies

$$\operatorname{mcc}(L_n) = \Omega\left(\frac{\operatorname{svnc}(L_n)^2}{\log_2 n}\right).$$

Thus there is an almost quadratic gap (and hence almost maximal gap) between Monte-Carlo communication and self-verifying nondeterminism. This consequence of Theorem 1.1, however, is superseded by a recent result of [12], which constructed a family of languages $L_n \subseteq \{0,1\}^n$ with two-sided error randomized communication complexity $\Theta(n)$ and self-verifying nondeterministic communication complexity $\Theta(\sqrt{n})$.

The proof of Theorem 1.1 utilizes the fact that a nondeterministic protocol with r advice bits is a collection of at most 2^r deterministic protocols. Hence one such deterministic protocol exists that accepts at least $\frac{|L_{n,t(n)}|}{2^r}$ entries of $L_{n,t(n)}$ and rejects all elements of the complement. Let us call such a protocol an r-protocol.

Proof techniques are required that give good results for the communication complexity of r-protocols but weak results for nondeterministic protocols. We briefly discuss the most prominent general techniques, i.e., the rank method, fooling sets, and upper-bounding the size of monochromatic submatrices.

The rank method does not seem to be adequate, since it does not apply to nondeterminism, even when limiting the number of advice bits. Moreover, the following observation shows that each language $L \subseteq \{0,1\}^n$ can be decomposed into $O(\log_2 n)$

languages L_i such that the rank of each L_i over $\mathbb{Z}_2$ is bounded by the nondeterministic communication complexity of L. The argument, however, does not apply to the rank over the reals. (For a language $K \subseteq \{0,1\}^{2n}$ let $\text{rank}_F(K)$ be the rank of the communication matrix of K over the field F.)

PROPOSITION 1.4. *Let $L \subseteq \{0,1\}^{2n}$ be an arbitrary language with $\text{nc}(L) = c$. Then languages $L_1, \ldots, L_{2n+1} \subseteq \{0,1\}^{2n}$ exist such that*

$$L = \bigcup_{i=1}^{2n+1} L_i \quad \textit{and} \quad \log_2(\text{rank}_{\mathbb{Z}_2}(L_i)) \leq \text{nc}(L) \quad \textit{for all } i.$$

In other words, $\log_2(2n+1)$ advice bits suffice to recognize L if we are allowed to replace deterministic protocols by "small-rank" protocols.

Proof. Let $\text{nc}(L) = k$. Then we obtain the representation $M = \bigcup_{i=1}^{2^k} M_i$ for the communication matrix M of L: An optimal nondeterministic protocol consists of at most 2^k accepting messages, which each contribute a rank-1 matrix M_i. For a vector α of 2^k zeroes and ones we define the matrix

$$M(\alpha) = \sum_{i=1}^{2^k} \alpha_i \cdot M_i$$

over $\mathbb{Z}_2$. Observe that $\text{rank}(M(\alpha)) \leq 2^k$ and that

$$M[i,j] = 0 \Rightarrow M(\alpha)[i,j] = 0.$$

On the other hand, if we choose the vector α at random, then

$$\text{prob}[M(\alpha)[i,j] = 0] = \frac{1}{2}$$

for any *positive* entry (i,j) (i.e., $M[i,j] = 1$).

Hence if we choose $2n+1$ vectors $\alpha^{(1)}, \ldots, \alpha^{(2n+1)}$ at random, we get

$$\text{prob}[M(\alpha^{(1)})[i,j] = 0 \wedge \cdots \wedge M(\alpha^{(2n+1)})[i,j] = 0] = \frac{1}{2^{2n+1}}$$

for any positive entry (i,j). Thus $\alpha^{(1)}, \ldots, \alpha^{(2n+1)}$ exist with

$$M(\alpha) = \bigcup_{i=1}^{2n+1} M(\alpha_i),$$

and the claim follows if we define L_i as the language with communication matrix $M(\alpha^{(i)})$. □

The methods of 1-*fooling sets* [9, 15] (i.e., determining large sets of 1-entries of the communication matrix with the fooling set property) and 1-*chromatic submatrices* [9, 15] (i.e., upper-bounding the size of the largest 1-chromatic submatrix) have to fail since they yield lower bounds for nondeterministic communication as well. The corresponding methods of 0-*fooling sets* and 0-*chromatic submatrices* cannot be applied without change, since elements of the language may be rejected. Thus, in order to prove lower bounds for protocols with a restricted degree of nondeterminism, we have to develop new proof techniques.

The situation is far easier if we consider a sublogarithmic number of advice bits. The language $\text{NID}_n = \{(x,y) : x, y \in \{0,1\}^n \text{ and } x \neq y\}$ benefits the most if many advice bits are available.

PROPOSITION 1.5.
(a) *For each language* $L \subseteq \{0,1\}^{2n}$ *and any* $r \in N$,

$$\mathrm{nc}_r(L) \geq \frac{\mathrm{c}(L)}{2^r}.$$

(b) $\frac{n}{2^r} \leq \mathrm{nc}_{r+1}(\mathrm{NID}_n) \leq \frac{n}{2^r} + r + 1$.

Proof. (a) A nondeterministic protocol with r advice bits and c exchanged bits can be simulated by a deterministic protocol with $2^r \cdot c$ exchanged bits.

(b) We describe a nondeterministic protocol for NID_n. The protocol partitions the n bits of processor A into at most $2^r + 1$ intervals of length $\lfloor \frac{n}{2^r} \rfloor$. Processor A guesses an interval (with at most $r+1$ advice bits) and communicates the interval and its bits. □

The paper is organized as follows. The next section introduces a disjointness language that will be hard for restricted nondeterminism. Section 3 introduces the crucial concept of simple and double witnesses and shows some basic properties. A proof sketch and a description of a new approach for distinguishing deterministic and nondeterministic communication complexity make up section 4. The argument is completed in sections 5 through 7. Conclusions and open problems are given in section 8.

2. A disjointness language. From now on we fix nonnegative integers m, n, and s (with $n \leq m$). We begin with a construction of the language family mentioned in Theorem 1.1.

DEFINITION 2.1.
(a) $\mathcal{P}_{m,n}$ *denotes the collection of all* n*-element subsets of* $\{1, \ldots, m\}$, *and*

$$\mathcal{P}_{m,n}^{(s)} = \times_{i=1}^{s} \mathcal{P}_{m,n}.$$

(b) *Let* L *be a subset of* $\{0,1\}^{2n}$. *The set*

$$L^{(s)} = \{(x_1, \ldots, x_s; y_1, \ldots, y_s) : \forall\, i,\ x_i, y_i \in \{0,1\}^n \text{ and } (x_i, y_i) \in L\}$$

is called the s*-fold direct sum of* L.
(c) $D_{m,n}$ *denotes the complement of the language of* set disjointness *when inputs are subsets of* $\{1, \ldots, m\}$ *of size* n, *i.e.,*

$$D_{m,n} = \{(x, y)\ :\ x, y \in \mathcal{P}_{m,n} \text{ and } x \cap y \neq \emptyset\}.$$

Theorem 1.1 is an immediate consequence of the following reformulation.

THEOREM 2.2. *Assume that* m *and* n *are sufficiently large with* $m \geq n^{32}$.
(a) $\mathrm{nc}(D_{m,n}^{(s)}) = O(s \cdot \log_2 m)$ *and* $\mathrm{nc}(\overline{D_{m,n}^{(s)}}) = O(n + \log_2(s \cdot \log_2 m))$.
(b) $\mathrm{nc}_{o(s)}(D_{m,n}^{(s)}) = \Omega(s \cdot n)$.

Obviously $\mathrm{nc}(D_{m,n}^{(s)}) = O(s \cdot \log_2 m)$ follows by having a nondeterministic protocol guess a common element s times. We exploit the following surprising result, which provides an efficient nondeterministic protocol to verify disjointness.

PROPOSITION 2.3 (Example 2.12 in [15]). *Assume that processors* A *and* B *receive* k*-element subsets* $x, y \subseteq \{1, \ldots, N\}$, *respectively. Then there is a nondeterministic protocol that verifies whether* x *and* y *are disjoint by exchanging at most* $O(k + \log\log N)$ *bits.*

As a consequence of Proposition 2.3 we obtain $\mathrm{nc}(\overline{D^{(s)}_{m,n}}) = O(n+\log_2(s\cdot\log_2 m))$, since guessing a pair of disjoint sets requires just $\log_2 s$ bits. Thus it remains to show part (b), and this is the topic of subsequent sections. Next we show that Theorem 1.1 is a consequence of Theorem 2.2.

Proof of Theorem 1.1. Input size k and the upper bound $t(k)$ on the number of advice bits are given. Choose n such that

$$k = \Theta\left(t(k)\cdot\log_2\binom{n^{32}}{n}\right).$$

Thus $k = \Theta(t(k)\cdot n\cdot\log_2 n)$ (resp., $n\cdot\log_2 n = \Theta(\frac{k}{t(k)})$), and hence $\log_2 n = \Theta(\log_2(n\cdot\log_2 n)) = \Theta(\log_2\frac{k}{t(k)})$. We set

$$L_{k,t(k)} = D^{t(k)}_{n^{32},n}.$$

We observe first that a nondeterministic protocol can recognize $L_{k,t(k)}$ by guessing common elements for each of the $t(k)$ sets. Hence $\mathrm{nc}(L_{k,t(k)}) = O(t(k)\cdot\log_2 n) = O(t(k)\cdot\log_2\frac{k}{t(k)})$.

To recognize the complement we guess one of the $t(k)$ sets and apply Proposition 2.3 to verify disjointness. Thus

$$\begin{aligned}\mathrm{nc}(\overline{L_{k,t(k)}}) &= O(n+\log_2\log_2 m+\log_2 t(k))\\ &= O\left(\frac{k}{t(k)\cdot\log_2 n}+\log_2 t(k)\right)\\ &= O\left(\frac{k}{t(k)\cdot\log_2\frac{k}{t(k)}}+\log_2 t(k)\right).\end{aligned}$$

Alternatively, we guess one of the $t(k)$ sets and then communicate this set completely. This gives

$$\begin{aligned}\mathrm{nc}_{\log_2 t(k)}(\overline{L_{k,t(k)}}) &= O(n\cdot\log_2 n+\log_2 t(k))\\ &= O\left(\frac{k}{t(k)}+\log_2 t(k)\right).\end{aligned}$$

We now use Theorem 2.2(b) for the lower bound on $L_{k,t(k)}$:

$$\begin{aligned}\mathrm{nc}_{o(t(k))}(L_{k,t(k)}) &= \mathrm{nc}_{o(t(k))}(D^{t(k)}_{n^{32},n})\\ &= \Omega(t(k)\cdot n) = \Omega\left(\frac{k}{\log_2 n}\right)\\ &= \Omega\left(\frac{k}{\log_2\frac{k}{t(k)}}\right),\end{aligned}$$

and the claim follows. □

3. Simple and double witnesses. Our argument is tailor-made for $D^{(s)}_{m,n}$ and more generally for direct sums of languages with small nondeterministic communication complexity. We utilize the properties of such languages through the crucial concepts of simple and double witnesses.

DEFINITION 3.1. *Let I be a subset of $\{1,\dots,s\}$.*
(a) *An element*

$$w \in \times_{i=1}^{s}(\mathcal{P}_{m,0} \cup \mathcal{P}_{m,1} \cup \mathcal{P}_{m,2})$$

is called a witness. If $|w_i| \le 1$ for all components i and $I = \{i : |w_i| = 1\}$, then w is called a simple *witness with core I. If $1 \le |w_i| \le 2$ for all components i and $I = \{i : |w_i| = 2\}$, then w is called a* double *witness with core I. Finally, we say that a witness w with core I is a witness for $x \in \mathcal{P}_{m,n}^{(s)}$, provided $w_i \subseteq x_i$ for all $i \in I$.*
(b) *$\mathrm{Witness}_{I,1}$ is the set of simple witnesses with core I, and $\mathrm{Witness}_{I,2}$ is the set of double witnesses with core I. We define for $r \in \{1,2\}$ and $x \in \mathcal{P}_{m,n}^{(s)}$*

$$\mathrm{Witness}_{I,r}(x) = \{w \ : \ w \in \mathrm{Witness}_{I,r} \text{ and for all } i \in I,\ w_i \subseteq x_i \ \}.$$

(c) *Let w be a witness, and let X be a subset of $\mathcal{P}_{m,n}^{(s)}$. Then*

$$p_X(w) = \frac{|\{x \in X \ : \ \text{for all } i,\ w_i \subseteq x_i \ \}|}{|X|}$$

is the probability that a random element of X possesses w as a witness.

If two elements $x, y \in \mathcal{P}_{m,n}^{(s)}$ possess a common simple witness w (with core I), then this witness guarantees an intersection of x_i and y_i for every component $i \in I$. Thus any common witness w with core $\{1,\dots,s\}$ testifies that (x,y) belongs to $D_{m,n}^{(s)}$.

We show in the next lemma that there is a *small* set of witnesses that is able to testify for a *large* number of entries of a 1-chromatic submatrix. Hence, on an intuitive level, the canonical nondeterministic protocol that guesses one witness *should* not be too far away from an optimal protocol.

LEMMA 3.2. *Let X and Y be subsets of $\mathcal{P}_{m,n}^{(s)}$ with $M = X \times Y$ as the induced submatrix of the communication matrix for $D_{m,n}^{(s)}$. Let I be a subset of $\{1,\dots,s\}$ of size k.*
(a) *$\sum_{w \in \mathrm{Witness}_{I,1}} p_X(w) = n^k$ and $\sum_{w \in \mathrm{Witness}_{I,1}} p_Y(w) = n^k$.*
(b) *Assume that M is 1-chromatic. Then*

$$\sum_{w \in \mathrm{Witness}_{I,1}} p_X(w) \cdot p_Y(w) \ge 1.$$

(c) *For $0 \le \rho \le 1$ set $W_I(\rho) = \{w \in \mathrm{Witness}_{I,1} : p_X(w) \ge \rho \text{ and } p_Y(w) \ge \rho\}$. Assume that M is 1-chromatic. Then, for every $\alpha > 0$,*

$$|W_I(\alpha \cdot n^{-k})| \le \frac{n^{2k}}{\alpha}$$

and

$$\sum_{w \in W_I(\alpha \cdot n^{-k})} p_X(w) \cdot p_Y(w) \ge 1 - 2\alpha.$$

Remark 3.1. Fix the core I of size k. Observe that there are exactly m^k simple witnesses with core I. Therefore, as a consequence of part (a), $(\frac{n}{m})^k$ is the average probability for a random element in X having a given witness. Hence the set $W_I(\alpha \cdot n^{-k})$ consists of highly likely witnesses, since $m \ge n^{32}$.

For $\alpha = \frac{1}{4}$ we get relatively few, namely $4 \cdot n^{2k} \ll m^k$, simple witnesses that already contribute $\frac{1}{2}$ towards the sum $\sum_{w \in \text{Witness}_{I,1}} p_X(w) \cdot p_Y(w)$.

Proof. (a) We first observe that $|\text{Witness}_{I,1}(x)| = n^k$ and hence obtain

$$\begin{aligned}\sum_{w \in \text{Witness}_{I,1}} p_X(w) &= \sum_{w \in \text{Witness}_{I,1}} \sum_{x \in X, w \in \text{Witness}_{I,1}(x)} \frac{1}{|X|} \\ &= \sum_{x \in X} \sum_{w \in \text{Witness}_{I,1}(x)} \frac{1}{|X|} \\ &= \sum_{x \in X} \frac{|\text{Witness}_{I,1}(x)|}{|X|} = \sum_{x \in X} \frac{n^k}{|X|} = n^k.\end{aligned}$$

(b) Assume that M is 1-chromatic. Then for each entry (x, y) of M there exists a simple witness w with core I and $w_i \subseteq x_i \cap y_i$ for all $i \in I$. Hence, since $p_X(w) \cdot |X|$ (resp., $p_Y(w) \cdot |Y|$) is the number of rows (resp., columns) with witness w,

$$|X| \cdot |Y| \leq \sum_{w \in \text{Witness}_{I,1}} p_X(w)|X| \cdot p_Y(w)|Y|,$$

and the claim follows.

(c) The upper bound on $|W_I(\alpha \cdot n^{-k})|$ is an immediate consequence of part (a). Moreover, we obtain again with part (a) that

$$\begin{aligned}&\sum_{w \in \text{Witness}_{I,1},\ w \notin W_I(\alpha \cdot n^{-k})} p_X(w) \cdot p_Y(w) \\ &\quad \leq \sum_{w \in \text{Witness}_{I,1}} p_X(w) \cdot \alpha \cdot n^{-k} + \sum_{w \in \text{Witness}_{I,1}} \alpha \cdot n^{-k} \cdot p_Y(w) \leq 2\alpha,\end{aligned}$$

and the remaining part of the claim follows. □

Next, we determine the size of $D^{(s)}_{m,n}$ and the expected number of entries with a double witness. It should not come as a surprise that this expected number is small when compared to the size of $D^{(s)}_{m,n}$, since double intersections are rare.

LEMMA 3.3.

(a) *If m and n are sufficiently large with $m \geq n^{32}$, then*

$$|D^{(s)}_{m,n}| \geq e^{-s/n} \cdot \left(\frac{n^2}{m}\right)^s \cdot \binom{m}{n}^{2s}.$$

(b) *Let $X = \mathcal{P}^{(s)}_{m,n}$. Then for any integer k $(1 \leq k \leq s)$*

$$\sum_{I \subseteq \{1,\ldots,s\}, |I|=k} \sum_{w \in \text{Witness}_{I,2}} p_X(w) \cdot p_X(w) \leq 2^{-k} \cdot \binom{s}{k} \cdot \left(\frac{n^2}{m}\right)^{s+k}.$$

Proof. (a) Fix an element $x \in \mathcal{P}^{(1)}_{m,n}$. If a set $y \in \mathcal{P}^{(1)}_{m,n}$ is randomly chosen, then

$$\text{prob}[x \text{ and } y \text{ intersect}] \geq n \cdot \frac{\binom{m-n}{n-1}}{\binom{m}{n}},$$

since the right-hand side counts intersections in exactly one element. Hence, provided that m and n are sufficiently large with $m \geq n^{32}$,

$$\begin{aligned}
\text{prob}[x \text{ and } y \text{ intersect}] &\geq \frac{n^2}{m} \cdot \frac{\binom{m-n}{n-1}}{\binom{m-1}{n-1}} \\
&= \frac{n^2}{m} \cdot \frac{m-n}{m-1} \cdots \frac{m-2n+2}{m-n+1} \\
&= \frac{n^2}{m} \cdot \left(1 - \frac{n-1}{m-1}\right) \cdots \left(1 - \frac{n-1}{m-n+1}\right) \\
&\geq \frac{n^2}{m} \cdot \left(1 - \frac{n-1}{m-n+1}\right)^{n-1} \\
&\geq \frac{n^2}{m} \cdot e^{-(n-1)\cdot\frac{n-1}{m-2n+2}} \qquad \text{since } (1-x) \geq e^{\frac{-x}{1-x}} \\
&\geq \frac{n^2}{m} \cdot e^{-1/n}.
\end{aligned}$$

Thus $|D^{(1)}_{m,n}| \geq e^{-1/n} \cdot (\frac{n^2}{m}) \cdot \binom{m}{n}^2$, and the claim follows since $|D^{(s)}_{m,n}| = |D^{(1)}_{m,n}|^s$.

(b) Let w be a double witness with core I and $|I| = k$. Remember that $X = \mathcal{P}^{(s)}_{m,n}$. Then

$$\begin{aligned}
p_X(w) &= \frac{\binom{m-2}{n-2}^k}{\binom{m}{n}^k} \cdot \frac{\binom{m-1}{n-1}^{s-k}}{\binom{m}{n}^{s-k}} \\
&= \left(\frac{n \cdot (n-1)}{m \cdot (m-1)}\right)^k \cdot \left(\frac{n}{m}\right)^{s-k} \\
&\leq \left(\frac{n}{m}\right)^{2k} \cdot \left(\frac{n}{m}\right)^{s-k} = \left(\frac{n}{m}\right)^{s+k}.
\end{aligned}$$

Hence

$$\begin{aligned}
\sum_{w \in \text{Witness}_{I,2}} p_X(w) \cdot p_X(w) &\leq \binom{m}{2}^k \cdot m^{s-k} \cdot \left(\frac{n}{m}\right)^{s+k} \cdot \left(\frac{n}{m}\right)^{s+k} \\
&\leq 2^{-k} \cdot m^{2k} \cdot m^{s-k} \cdot \left(\frac{n}{m}\right)^{s+k} \cdot \left(\frac{n}{m}\right)^{s+k} \\
&= 2^{-k} \cdot \left(\frac{n^2}{m}\right)^{s+k},
\end{aligned}$$

and the claim follows. □

4. A proof sketch. We first give a rough proof outline of Theorem 2.2(b). Assume that P_N is a nondeterministic protocol, which accepts $D^{(s)}_{m,n}$ with r advice bits and communicates $o(s \cdot n)$ bits. Then P_N consists of a collection of at most 2^r deterministic protocols $P_1, \ldots, P_{2^r}$, where each P_i accepts only inputs that belong to $D^{(s)}_{m,n}$ and communicates at most $o(s \cdot n)$ bits.

In order to show that r advice bits are also necessary it suffices to show that any *deterministic* protocol Q, which accepts only inputs from $D^{(s)}_{m,n}$ and exchanges at

most $o(s \cdot n)$ bits, accepts fewer than $|D^{(s)}_{m,n}|/2^r$ elements from $D^{(s)}_{m,n}$. Assume to the contrary that Q accepts at least the fraction

$$2 \cdot \frac{|D^{(s)}_{m,n}|}{2^{\varepsilon \cdot s}}$$

of elements from $D^{(s)}_{m,n}$. Then at least half of the entries that are accepted by Q belong to large 1-chromatic matrices $M_1, \ldots, M_x$, each with at least $\binom{m}{n}^s / 2^{o(s \cdot n)}$ rows and columns. Assume that M_i has X_i (resp., Y_i) as its set of rows (resp., columns) and let $X_0 = Y_0 = \mathcal{P}^{(s)}_{m,n}$ be the set of rows and columns of the communication matrix. Since Q is a deterministic protocol we get

$$\sum_{I,|I|=k} \sum_{w \in \mathrm{Witness}_{I,2}} p_{X_0}(w) \cdot p_{Y_0}(w) \geq \sum_{i=1}^{x} \frac{|X_i| \cdot |Y_i|}{|X_0| \cdot |Y_0|} \cdot \sum_{I,|I|=k} \sum_{w \in \mathrm{Witness}_{I,2}} p_{X_i}(w) \cdot p_{Y_i}(w). \tag{4.1}$$

Now, according to Lemma 3.2, there is a small set W^* of simple witnesses (with core $\{1, \ldots, s\}$) which testify for most nonempty intersections of $M = M_i$. For such a witness $w \in W^*$ let $M(w)$ be the large submatrix of M consisting of all rows and columns of M with simple witness w.

Imagine that we remove w from the row and column vectors of $M(w)$. (In other words, if x is the label of a row or column of $M(w)$, then we replace x by x' with $x'_i = x_i - \{w_i\}$.) We obtain a large submatrix, which will, however, in general not be 1-chromatic. Now assume that *any* large, not necessarily 1-chromatic, matrix has a considerable number of simple witnesses that are quite likely for both rows and columns. Then, provided that M is large, $M(w)$ has a large number of simple witnesses, and these witnesses turn into double witnesses after merging with w. (In general we will not obtain many simple witnesses with core $I = \{1, \ldots, s\}$, since even a large submatrix may possess components i with $x_i \cap y_i = \emptyset$ for all of its rows x and all of its columns y. Hence we have to work with large core sets.)

The following lemma expresses that M will indeed have many double witnesses.

LEMMA 4.1. *Let $\varepsilon \in]0,1[$ be sufficiently small. Assume that m and n are sufficiently large with $m \geq n^{32}$. Let $M = X \times Y$ be a 1-chromatic submatrix with*

$$|X| \geq 4 \cdot n^s \cdot \alpha_1^{s \cdot n} \cdot \binom{m}{n}^s \quad \text{and} \quad |Y| \geq 4 \cdot n^s \cdot \alpha_2^{s \cdot n} \cdot \binom{m}{n}^s,$$

where $\alpha_1, \alpha_2 \in [0,1[$ are sufficiently large. Then, for $k = (1 - 2\varepsilon) \cdot s$,

$$\sum_{I,|I|=k} \sum_{w \in \mathrm{Witness}_{I,2}} p_X(w) \cdot p_Y(w) \geq 2^{-15k/16} \cdot \binom{s}{k} \cdot \left(\frac{n^2}{m}\right)^k.$$

Our goal is to show that Q produces, through its large disjoint 1-chromatic submatrices M_i, far too many double witnesses. We apply Lemma 4.1 and get with (4.1)

$$\begin{aligned} \sum_{I,|I|=k} \sum_{w \in \mathrm{Witness}_{I,2}} p_{X_0}(w) \cdot p_{Y_0}(w) &\geq \sum_{i=1}^{x} \frac{|X_i| \cdot |Y_i|}{|X_0| \cdot |Y_0|} \cdot 2^{-15k/16} \cdot \binom{s}{k} \cdot \left(\frac{n^2}{m}\right)^k \\ &\geq \frac{|D^{(s)}_{m,n}|}{2^{\varepsilon \cdot s} \cdot \binom{m}{n}^{2s}} \cdot 2^{-15k/16} \cdot \binom{s}{k} \cdot \left(\frac{n^2}{m}\right)^k. \end{aligned}$$

However,

$$\frac{|D^{(s)}_{m,n}|}{\binom{m}{n}^{2s}} \geq e^{-s/n} \cdot \left(\frac{n^2}{m}\right)^s$$

and therefore

$$\sum_{I,|I|=k} \sum_{w \in \mathrm{Witness}_{I,2}} p_{X_0}(w) \cdot p_{Y_0}(w) \geq e^{-s/n} \cdot 2^{-15k/16-\varepsilon \cdot s} \cdot \binom{s}{k} \cdot \left(\frac{n^2}{m}\right)^{s+k}$$

for $X_0 = Y_0 = \mathcal{P}^{(s)}_{m,n}$. This contradicts assertion (b) of Lemma 3.3, and Theorem 2.2 follows.

Thus Lemma 4.1 is the technical core of our argument. To obtain the rather precise bounds, we follow a qualitative approach; namely, we show that it suffices to consider monotone submatrices M. The next section treats the problem of multiple counting when determining the number of double witnesses. Monotone submatrices will be investigated in section 6. Finally we conclude the argument in section 7, where we show that *any* large, not necessarily 1-chromatic matrix has a considerable number of simple witnesses which are quite likely for both rows and columns.

5. 1-chromatic submatrices and double witnesses. Our goal is to show Lemma 4.1. In a first step we reduce the problem of counting double witnesses in large 1-chromatic submatrices to the problem of counting simple witnesses in large but otherwise arbitrary submatrices.

We fix a 1-chromatic matrix $M = X \times Y$ with the properties described in Lemma 4.1. With Lemma 3.2(c) we obtain (for $\alpha = \frac{1}{4}$) a set W^* of simple witnesses with core $\{1, \ldots, s\}$ such that

$$p_X(w), p_Y(w) \geq \frac{1}{4 \cdot n^s} \quad \text{for all } w \in W^*, \tag{5.1}$$

$$|W^*| \leq 4 \cdot n^{2s}, \quad \text{and} \tag{5.2}$$

$$\sum_{w \in W^*} p_X(w) \cdot p_Y(w) \geq \frac{1}{2}. \tag{5.3}$$

Let ε be a positive real number to be determined later. For the moment we fix a subset $I \subseteq \{1, \ldots, s\}$ of size $k = (1 - 2\varepsilon) \cdot s \geq s/2$. For each $w^* \in W^*$ let $M(w^*) = X(w^*) \times Y(w^*)$ be the submatrix of $M = X \times Y$ consisting of exactly those entries (x, y) of M with w^* as a common witness (i.e., $w^*_i \subseteq x_i \cap y_i$ for all i). We obtain for each simple witness $w \in \mathrm{Witness}_{I,1}$ (with $w_i \neq w^*_i$ for all elements $i \in I$) the double witness $u = w^* \cup w$ for $M(w^*)$; i.e., $u_i = w^*_i \cup w_i$ for all i. However, we cannot immediately infer that the matrix M has a correspondingly large number of double witnesses with core I, since the double witness u may be induced by too many simple witnesses. Observe that this can happen only if u contains too many witnesses from the set W^*.

Let $u \in \mathrm{Witness}_{I,2}$ be a double witness. The set

$$W^*(u) := \{ w^* \in W^* \ : \ w^*_i \subseteq u_i \text{ for all } i \in I \}$$

contains all witnesses from W^* that "generate" u. We say that the simple witness $w \in \mathrm{Witness}_{I,1}$ *counts* if there is no double witness $u \in \mathrm{Witness}_{I,2}$ with $w_i \subseteq u_i$ for

all $i \in I$ and $|W^*(u)| \geq 2^{7k/8}$. If a witness counts, then none of its generated double witnesses contains more than $2^{7k/8}$ witnesses from W^*, and consequently multiple counting will be "modest." How many simple witnesses do not count?

Assume that w does not count and that the double witness u is the reason. If all elements of $W^*(u)$ agree on at most $k/4$ components from I with w, then they agree with $u - w$ on the remaining components, and hence

$$\begin{aligned}
|W^*(u)| &\leq \sum_{i=0}^{k/4} \binom{k}{i} \leq k \cdot \binom{k}{k/4} \\
&\leq k \cdot 4^{k/4} \cdot \left(\frac{4}{3}\right)^{3k/4} \qquad \text{since } \binom{t}{\alpha \cdot t} \leq \left(\frac{1}{\alpha}\right)^{\alpha \cdot t} \cdot \left(\frac{1}{1-\alpha}\right)^{(1-\alpha)\cdot t} \\
&= k \cdot \left(4 \cdot \frac{4^3}{3^3}\right)^{k/4} = k \cdot \left(\frac{4}{3^{3/4}}\right)^k \\
&< 2^{7k/8}.
\end{aligned}$$

The last inequality holds for sufficiently large k, since $\frac{4}{3^{3/4}} < 2^{7/8}$. In other words, if w does not count, then some element $w^* \in W^*$ agrees with w on at least $k/4$ components in I. But then, for $n \leq m^{1/32}$ and m sufficiently large, at most

$$\begin{aligned}
|W^*| \cdot \binom{k}{k/4} \cdot m^{3k/4} &\leq 4 \cdot n^{2s} \cdot \binom{k}{k/4} \cdot m^{3k/4} \qquad \text{because of (5.2)} \\
&\leq 2^{s+2} \cdot \left(\frac{n^{2s}}{m^{k/4}}\right) \cdot m^k \qquad \text{since } \binom{k}{k/4} \leq 2^s \\
&\leq 2^{s+2} \cdot \frac{m^k}{m^{k/8}} \qquad \text{since } n^2 \leq m^{1/16} \leq m^{k/(8\cdot s)} \\
&\leq (\varepsilon_2 \cdot m)^{(1-11\varepsilon_2)k}, \quad \text{where } \varepsilon_2 = \frac{1}{99}, \qquad (5.4)
\end{aligned}$$

witnesses in $\text{Witness}_{I,1}$ do not count.

The following lemma provides the promised reduction to counting simple witnesses. Before presenting the lemma, we set up a number of constants. Let w^* be an element of W^*. We decrement m and n by one to reflect that only restrictions of witnesses $w \in \times_{i=1}^{s}\{1,\ldots,m\} - \{w_i^*\}$ are considered. Remember that $|X| = 4n^s \cdot \alpha_1^{sn} \cdot \binom{m}{n}^s$, by assumption on the matrix M. Since $p_X(w^*) \geq \frac{1}{4\cdot n^s}$ (with (5.1)), we get

$$|X(w^*)| \geq \alpha_1^{s\cdot n} \cdot \binom{m}{n}^s \geq \alpha_1^{s\cdot(n-1)} \cdot \binom{m-1}{n-1}^s,$$

where the last inequality holds for sufficiently large m and n. The analogous statement holds of course for $|Y(w^*)|$ as well. Moreover, we choose $\alpha_1, \alpha_2 < 1$ so large that the following hold, with $\varepsilon_1 = \varepsilon$ and $\varepsilon_2 = \frac{1}{80}$:

$$\begin{aligned}
\alpha_1, \alpha_2 &> 4^{1/n-1} \cdot e^{-\varepsilon_1^4/2} \quad \text{and} \quad \varepsilon_1^2 \geq \frac{2(n-1)}{m-1}, \\
\alpha_1, \alpha_2 &> 4^{1/(n-1)} \cdot e^{-\varepsilon_2^3\cdot(1-\varepsilon_2)/4} \quad \text{and} \quad \varepsilon_2 \geq \frac{n-1}{(1-\varepsilon_2)\cdot(m-1)}, \\
\frac{1}{32} \cdot \frac{(1-\varepsilon_1)^{3s}}{\binom{s}{\varepsilon_1 \cdot s}} &\geq e^{-\varepsilon_2^2 \cdot s/2}.
\end{aligned}$$

This is clearly possible for the first four inequalities, provided m and n are sufficiently

large and $n^{32} \le m$. The fifth inequality follows if $\varepsilon_1 = \varepsilon$ is sufficiently small.

LEMMA 5.1. *$\alpha_1, \alpha_2 \in [0,1]$ as well as $\varepsilon_1, \varepsilon_2 \in [0, 1/6]$ satisfy the inequalities*

$$\alpha_1, \alpha_2 > 4^{1/n} \cdot e^{-\varepsilon_1^4/2} \quad \text{and} \quad \varepsilon_1^2 \ge \frac{2n}{m}, \tag{5.5}$$

$$\alpha_1, \alpha_2 > 4^{1/n} \cdot e^{-\varepsilon_2^3 \cdot (1-\varepsilon_2)/4} \quad \text{and} \quad \varepsilon_2 \ge \frac{n}{(1-\varepsilon_2)\cdot m}, \tag{5.6}$$

$$\frac{1}{32} \cdot \frac{(1-\varepsilon_1)^{3s}}{\binom{s}{\varepsilon_1 \cdot s}} \ge e^{-3 \cdot \varepsilon_2 \cdot s/2}. \tag{5.7}$$

Assume that X and Y are subsets of $\mathcal{P}^{(s)}_{m,n}$, with $|X| = \alpha_1^{s\cdot n} \cdot \binom{m}{n}^s$ and $|Y| = \alpha_2^{s\cdot n} \cdot \binom{m}{n}^s$. Then there is a subset $I \subseteq \{1, \ldots, s\}$ of size at least $k = (1-2\varepsilon_1)\cdot s$ such that, for any set W of simple witnesses with core I,

$$\sum_{w \in \mathrm{Witness}_{I,1} - W} p_X(w) \cdot p_Y(w) \ge \frac{1}{32} \cdot \frac{(1-\varepsilon_1)^{5k}}{\binom{s}{\varepsilon_1 \cdot s}^2} \cdot \left(\frac{n^2}{m}\right)^k,$$

provided $|W| \le (\varepsilon_2 \cdot m)^{(1-11\cdot\varepsilon_2)\cdot s}$.

In what follows we show that Lemma 5.1 implies Lemma 4.1. The proof of Lemma 5.1 is given in the subsequent sections.

We apply Lemma 5.1 to the submatrix $X(w^*) \times Y(w^*)$ and obtain a subset $I(w^*) \subseteq \{1, \ldots, s\}$ of size $k = (1-2\varepsilon)\cdot s$. Observe that at most $(\varepsilon_2 \cdot m)^{(1-11\varepsilon_2)k}$ simple witnesses from $\mathrm{Witness}_{I(w^*),1}$ do not count (because of (5.4)), and hence

$$\begin{aligned}\sum_{w \in \mathrm{Witness}_{I(w^*)} \text{ and } w \text{ counts}} p_{X(w^*)}(w) \cdot p_{Y(w^*)}(w) &\ge \frac{1}{32} \cdot \frac{(1-\varepsilon)^{5k}}{\binom{s}{\varepsilon\cdot s}^2} \cdot \left(\frac{(n-1)^2}{m-1}\right)^k \\ &\ge \frac{1}{64} \cdot \frac{(1-\varepsilon)^{5k}}{\binom{s}{\varepsilon\cdot s}^2} \cdot \left(\frac{n^2}{m}\right)^k.\end{aligned}$$

The last inequality holds for $n \ge 4$. We set

$$\mathrm{simple}(w^*, I) := \sum_{w \in \mathrm{Witness}_{I,1} \text{ and } w \text{ counts}} p_{X(w^*)}(w) \cdot p_{Y(w^*)}(w).$$

If a witness counts, then none of its generated double witnesses contains more than $2^{7k/8}$ witnesses from W^*. Hence

$$\begin{aligned}&\sum_{|I|=k} \sum_{w \in \mathrm{Witness}_{I,2}} p_X(w) \cdot p_Y(w) \\ &\quad\ge 2^{-7k/8} \cdot \sum_{|I|=k} \sum_{w^* \in W^*} p_X(w^*) \cdot p_Y(w^*) \cdot \mathrm{simple}(w^*, I) \\ &\quad\ge 2^{-7k/8} \cdot \sum_{w^* \in W^*} p_X(w^*) \cdot p_Y(w^*) \cdot \mathrm{simple}(w^*, I(w^*)) \\ &\quad\ge 2^{-7k/8} \cdot \sum_{w^* \in W^*} p_X(w^*) \cdot p_Y(w^*) \cdot \frac{1}{64} \cdot \frac{(1-\varepsilon)^{5k}}{\binom{s}{\varepsilon\cdot s}^2} \cdot \left(\frac{n^2}{m}\right)^k \\ &\quad\ge 2^{-7k/8} \cdot \frac{1}{128} \cdot \frac{(1-\varepsilon)^{5k}}{\binom{s}{\varepsilon\cdot s}^2} \cdot \left(\frac{n^2}{m}\right)^k \qquad \text{with (5.3)} \\ &\quad\ge 2^{-15k/16} \cdot \binom{s}{k} \cdot \left(\frac{n^2}{m}\right)^k,\end{aligned}$$

provided that the positive real number ε is chosen sufficiently small (and hence k is sufficiently large).

This concludes the proof of Lemma 4.1. We still have to deal with the problem of counting simple witnesses when disregarding a small witness-set. We start to tackle this problem in the next section, where we show that it suffices to investigate monotone submatrices.

6. Monotone submatrices. We have to give rather precise estimates of the number of entries with a double witness for large 1-chromatic submatrices. As already outlined above, we start by estimating the number of entries with a simple witness for large (not necessarily 1-chromatic) submatrices. To obtain tight bounds we follow a qualitative approach; namely, we show that it suffices to consider monotone submatrices.

DEFINITION 6.1.

(a) *Let* $x = (x_1, \ldots, x_s)$ *and* $y = (y_1, \ldots, y_s)$ *be elements of* $\mathcal{P}^{(s)}_{m,n}$ *with* $x_i = \{x_{i,1} < \cdots < x_{i,n}\}$ *and* $y_i = \{y_{i,1} < \cdots < y_{i,n}\}$. *Then*

$$x \leq y \quad \Leftrightarrow \quad x_{i,j} \leq y_{i,j} \quad \text{for all } i, j.$$

(b) *A subset* $X \subseteq \mathcal{P}^{(s)}_{m,n}$ *is called monotone decreasing (resp., monotone increasing) if for all* $x \in X$ *and for all* $y \in \mathcal{P}^{(s)}_{m,n}$

$$y \in X \text{ whenever } y \leq x \text{ (resp., } y \in X \text{ whenever } x \leq y).$$

(c) *A matrix* $M = X \times Y$ *is monotone provided* X *is monotone decreasing and* Y *is monotone increasing.*

Our first goal is to show that monotone submatrices minimize the sum

$$\sum_{w \in \mathrm{Witness}_{I,1}} p_X(w) \cdot p_Y(w).$$

This can be verified along the lines of Frankl and Füredi [8] (establishing Harper's isoperimetric inequality for the hypercube). In particular, they show that the minimal distance between two sets of vertices of the n-dimensional hypercube is minimized by choosing (generalized) Hamming balls centered at 0 (resp., 1). Since a Hamming ball around 0 (resp., 1) is in our terminology monotone decreasing (resp., increasing), this result supports the intuition that monotone submatrices "minimize intersections."

However, the situation gets involved since we also have to deal with a set W of forbidden witnesses and the transformation from an arbitrary matrix to a monotone matrix loses track of those witnesses. Therefore we introduce the concept of a *system*, consisting of a submatrix $M = X \times Y$ and assignments f, g of sets of forbidden witnesses to elements of X and Y, respectively.

DEFINITION 6.2. *Let* I *be a subset of* $\{1, \ldots, s\}$. *A* system (X, f, Y, g, I) *consists of subsets* $X, Y \subseteq \mathcal{P}^{(s)}_{m,n}$ *and assignments* $f : X \to \mathcal{P}(\mathrm{Witness}_{I,1})$ *and* $g : Y \to \mathcal{P}(\mathrm{Witness}_{I,1})$. *We define*

$$\mathrm{legal}_I(X, f, Y, g) = \{(x, w, y) \,:\, x \in X, y \in Y, \\ w \in (\mathrm{Witness}_{I,1}(x) - f(x)) \cap (\mathrm{Witness}_{I,1}(y) - g(y))\}.$$

Moreover, we demand that $f(x) \subseteq \mathrm{Witness}_{I,1}(x)$ *and* $g(y) \subseteq \mathrm{Witness}_{I,1}(y)$ *for all* $x, y \in Y$.

The next lemma allows us to restrict attention to monotone submatrices only.

LEMMA 6.3. *The sets $X, Y \subseteq \mathcal{P}^{(s)}_{m,n}$ are given. Then there is a monotone decreasing subset X' and a monotone increasing subset Y' with*

$$|X| = |X'| \quad \textit{and} \quad |Y| = |Y'|.$$

Moreover, for any subset $I \subseteq \{1, \ldots, s\}$ and any system (X, f, Y, g, I), a system (X', f', Y', g', I) exists such that

$$\sum_{x \in X} |f(x)| = \sum_{x' \in X'} |f'(x')| \quad \textit{and} \quad \sum_{y \in Y} |g(y)| = \sum_{y' \in Y'} |g'(y')|, \tag{6.1}$$

as well as

$$|\operatorname{legal}_I(X, f, Y, g)| \geq |\operatorname{legal}_I(X', f', Y', g')|. \tag{6.2}$$

Remark 6.1. Observe that the transformation from the sets X and Y to the sets X' and Y' is independent of the assignments f, g and the core I.

Let $p^*_X(w)$ be the probability that w occurs as a *legal* witness for a row of X, and define $p^*_Y(w)$, $p^*_{X'}(w)$, and $p^*_{Y'}(w)$ analogously. The last inequality of Lemma 6.3 now implies that

$$\sum_{w \in \operatorname{Witness}_{I,1}} p^*_X(w) \cdot p^*_Y(w) \geq \sum_{w \in \operatorname{Witness}_{I,1}} p^*_{X'}(w) \cdot p^*_{Y'}(w),$$

and we have reached our goal. However, now a witness may be legal for some rows and columns and forbidden for others. However, the number of pairs (x, w)—with w forbidden for x—remains unchanged.

Proof. We initially follow the construction in the argument of Frankl and Füredi [8]. In particular, set

$$[z] = \sum_{i=1}^{s} \sum_{j=1}^{n} z_{i,j}$$

for $z \in \mathcal{P}^{(s)}_{m,n}$, and introduce the weight-function

$$\operatorname{weight}(Z) = \sum_{z \in Z} [z]$$

for a subset $Z \subseteq \mathcal{P}^{(s)}_{m,n}$. If X is not monotone decreasing or if Y is not monotone increasing, then we construct subsets X' and Y' of $\mathcal{P}^{(s)}_{m,n}$ such that properties (6.1), (6.2) of the lemma are satisfied and

$$\operatorname{weight}(Y') - \operatorname{weight}(X') > \operatorname{weight}(Y) - \operatorname{weight}(X).$$

Obviously the lemma follows after repeating this procedure until monotonicity is reached.

We assume without loss of generality that X is not monotone decreasing and, in particular, that there is $x = (x_1, x_2, \ldots, x_s) \in X$ with $\xi \in x_1$, $\xi - 1 \notin x_1$ such that

$$(x_1 - \{\xi\} \cup \{\xi - 1\}, x_2, \ldots, x_s) \notin X.$$

For $x \in X$ set $\hat{x} = (x_1 - \{\xi\} \cup \{\xi - 1\}, x_2, \ldots, x_s)$ and define

$$\text{decrease}_\xi(x) = \begin{cases} \hat{x} & \text{if } x_1 \cap \{\xi - 1, \xi\} = \{\xi\} \text{ and } \hat{x} \notin X, \\ x & \text{otherwise.} \end{cases}$$

For $y \in Y$ set $\hat{y} = (y_1 - \{\xi - 1\} \cup \{\xi\}, y_2, \ldots, y_s)$ and define

$$\text{increase}_\xi(y) = \begin{cases} \hat{y} & \text{if } y_1 \cap \{\xi - 1, \xi\} = \{\xi - 1\} \text{ and } \hat{y} \notin Y, \\ y & \text{otherwise.} \end{cases}$$

We replace X and Y by

$$X' = \{\text{decrease}_\xi(x) \mid x \in X\} \quad \text{and} \quad Y' = \{\text{increase}_\xi(y) \in Y \mid y \in Y\}.$$

Thus we move x "down" (resp., y "up") whenever possible. We say that $x^* \in X$ is a *twin* of $x \in X$ if x^* and x differ only in their first component and one obtains x_1^* from x_1 by replacing ξ by $\xi - 1$ or vice versa. If $\{\xi - 1, \xi\} \subseteq x_1$, then x is called its own twin. Observe that a twin prevents x from moving down.

We get $|X| = |X'|$, $|Y| = |Y'|$, and moreover $\text{weight}(Y') - \text{weight}(X') > \text{weight}(Y) - \text{weight}(X)$, since $\text{weight}(Y') \geq \text{weight}(Y)$ and $\text{weight}(X') < \text{weight}(X)$.

Assume that $x' \in X'$ and $y' \in Y'$ are given with $\text{decrease}_\xi(x) = x'$ and $\text{increase}_\xi(y) = y'$ for $x \in X$ and $y \in Y$. We first consider the easy case $1 \notin I$ and define $f'(x') = f(x)$ and $g'(y') = g(y)$. Obviously property (6.1) holds as well as property (6.2): If $w \in \text{Witness}_{I,1}(x) - f(x)$, then $w \in \text{Witness}_{I,1}(x') - f'(x')$ since $1 \notin I$.

The case $1 \in I$ has to be treated more carefully. In particular, we move up forbidden witnesses for x' whenever possible. Why? Property (6.2) requires that the new system have no more legal triples (x', w, y') than the old system. The decrease-operation for X moves sets down, whereas the increase-operation for Y moves sets up, and thus the sets in X and Y are moved away from each other. Legal witnesses for the new sets x' and y' should also move away from each other and hence should be moved down (resp., up). To create space for these movements we move forbidden witnesses in the opposite direction whenever possible; these movements are harmless, since the intersections created by forbidden witnesses are not counted.

We now define $f'(x')$ and $g'(y')$ with the help of the decrease and increase operations relative to $f(x)$ as opposed to X (resp., relative to $g(y)$ as opposed to Y).

Case 1. $x' \notin X$ (resp., $y' \notin Y$). Remember that we obtain x' (resp., y') by moving x down (resp., by moving y up). We set

$$f'(x') = \{\text{decrease}_\xi(w) \ : \ w \in f(x)\} \quad \text{and} \quad g'(y') = \{\text{increase}_\xi(w) \ : \ w \in g(y)\}.$$

Hence $f(x)$ (resp., $g(y)$) is moved down (resp., up) as well. From now on we omit the definition of g', since it will be completely analogous to the definition of f'.

Case 2. $x' \in X$. Observe that $x' = x$.

Case 2.1. x has no twin in X. We set $f'(x) = f(x)$.

Case 2.2. x has a twin in X. We assume without loss of generality that $\xi - 1 \in x_1$. This time f' will move up witnesses whenever possible. In particular, if $x \neq \text{twin}(x)$, then we redistribute forbidden witnesses within $f(x) \cup f(\text{twin}(x))$. The witnesses in $f(x)$ all have $\xi - 1$ as their element. We remove such a witness w from $f(x)$ and insert its twin w^* into $f(\text{twin}(x))$ iff w^* is not already an element of $f(\text{twin}(x))$. Hence we set

$$f'(x) = \{w \in f(x) : w_1 \neq \xi - 1\} \cup \{w \in f(x) : w_1 = \xi - 1 \text{ and } \text{twin}(w) \in f(\text{twin}(x))\}.$$

The remaining forbidden witnesses from the union $f(x) \cup f(\text{twin}(x))$ are placed into $f'(\text{twin}(x))$. We can move up forbidden witnesses also if $x = \text{twin}(x)$, i.e., if $\{\xi - 1, \xi\} \subseteq x_1$. We set

$$f'(x) = \{\text{increase}_\xi(w) \ : \ w \in f(x)\}.$$

Observe that our construction of f' and g' satisfies property (6.1) of Lemma 6.3: We have either $|f'(x')| = |f(x)|$ or $|f'(x')| + |f'(x^*)| = |f(x')| + |f(x^*)|$ for twins x' and x^*.

In order to verify property (6.2) we define an injection

$$\text{count} \ : \ \text{legal}_I(X', f', Y', g') \ \rightarrow \ \text{legal}_I(X, f, Y, g).$$

Let $(x', u', y') \in \text{legal}_I(X', f', Y', g')$ be given. We assume that $x' = \text{decrease}_\xi(x)$ and $y' = \text{increase}_\xi(y)$ for $x \in X$ and $y \in Y$. We set $u = u' - \{\xi\} \cup \{\xi - 1\}$.

Case 1. $u'_1 \notin \{\xi - 1, \xi\}$. Then $(x, u', y) \in \text{legal}_I(X, f, Y, g)$, and we set count$(x', u', y') = (x, u', y)$. Obviously count is injective when restricted to arguments satisfying the case assumption.

Case 2. $u'_1 = \xi$. Since $\xi \in x'_1$, we obtain $x' = x$, and x was not moved down due to the twin $x^* \in X$. Let y^* be the twin of y if y has a twin in Y. We first make a few easy observations.

(A) If $x = x^*$, then $f'(x')$ was obtained by applying the increase-operation to $f(x)$. But $u' \notin f'(x')$, since u' is legal and hence $u, u' \notin f(x)$.

(B) If $y = y^*$, then $g'(y')$ was obtained by applying the decrease-operation to $g(y)$. It is not possible that both u and u' belong to $g(y)$, since otherwise $u' \in g'(y)$.

(C) If $x \neq x^*$, then $f'(x')$ keeps all forbidden witnesses from $f(x)$ and receives increased witnesses from $f(x^*)$. Since $u' \notin f'(x)$, we have $u \notin f(x^*)$ and $u' \notin f(x)$.

(D) If $y \neq y^*$, then $g'(y')$ is the subset of $g(y)$ obtained by removing all witnesses w, whose decreased version does not belong to $g(y^*)$. But $u' \notin g'(y)$, and thus if $u' \in g(y)$, then $u \notin g(y^*)$.

Case 2.1. $y' \neq y$. Hence y was moved up and obviously $u_1 \in y_1$. If $x = x^*$, then $u \notin f(x) = f(x^*)$ with (A). If $x \neq x^*$, then $u \notin f(x^*)$ with (C). In either case $u \notin f(x^*)$ and $u_1 = \xi - 1 \in x^*_1$. We set count$(x', u', y') = (x^*, u, y)$.

We can assume $y' = y$ from now on.

Case 2.2. $y = y^*$ and $x = x^*$. With (B) one of u or u' does not belong to $g(y)$ and with (A) $u, u' \notin f(x)$. Thus, if $u \notin g(y)$, set count$(x', u', y') = (x, u, y)$, and otherwise set count$(x', u', y') = (x, u', y)$.

Case 2.3. $y \neq y^*$ and $x = x^*$. We have $u, u' \notin f(x)$ with (A). If $u \notin g(y^*)$, we set count$(x', u', y') = (x, u, y^*)$. Otherwise $u' \notin g(y)$ with (D) and we set count(x', u', y') $= (x, u', y)$.

Case 2.4. $y = y^*$ and $x \neq x^*$. We have $u \notin f(x^*)$ and $u' \notin f(x)$ with (C). Moreover, it is not possible that both u and u' belong to $f(y)$ with (B). If $u \notin f(y)$, then set count$(x', u', y') = (x^*, u, y)$. Otherwise $u' \notin g(y)$ and we set count$(x', u', y') =$ (x, u', y).

Case 2.5. $y \neq y^*$ and $x \neq x^*$. Again we have $u \notin f(x^*)$ and $u' \notin f(x)$ with (C). If $u \notin g(y^*)$, then we set count$(x', u', y') = (x^*, u, y^*)$. Otherwise $u' \notin g(y)$ with (D) and we set count$(x', u', y') = (x, u', y)$.

Case 3. $u'_1 = \xi - 1$. This case is analogous to Case 2.

Why is "count" injective? Assume that $\text{count}^{-1}(x,w,y)$ is nonempty. If $w_1 \notin \{\xi-1,\xi\}$, then (x,w,y) is generated through Case 1, but count, when restricted to Case 1, is injective.

Now assume that (x,w,y) is generated through Case 2. We observe by inspection that if (x,w,y) is generated by one subcase, then no other subcase can generate (x,w,y): If $y \neq y'$, then Case 2.1 is responsible and all other subcases can be uniquely identified when checking which of x and y is its own twin.

By symmetry, injectivity within Case 3 is guaranteed as well. Thus injectivity can only be violated if (x,w,y) is generated through Case 2 *and* Case 3. By checking for $x \neq x'$, $y \neq y'$, and which of x and y are its own twin, one finds that a collision is possible only if Case 2.i collides with 3.i for $i \geq 2$. Each Case 2.i attempts first to assign the witness u with $u_1 = \xi - 1$ and assigns u' with $u'_1 = \xi$ only when this is impossible. Case 3 follows this approach symmetrically by trying first to assign the witness v with $v_1 = \xi$ and assigns v' with $v'_1 = \xi - 1$ only when this is impossible.

Now it suffices to observe that a failure in the first attempt in Case 2.i makes a generation by Case 3.i impossible. □

The next proposition states a basic property of monotone sets, namely, that the probability of a witness increases if we reduce one or several of its components.

PROPOSITION 6.4. *Let X be a monotone decreasing subset of $\mathcal{P}^{(s)}_{m,n}$. Then for any subset $I \subseteq \{1,\ldots,s\}$ and for any simple witnesses $u,v \in \text{Witness}_{I,1}$ with $u_i \leq v_i$ for all $i \in I$,*

$$p_X(u) \geq p_X(v).$$

Proof. Let $I \subseteq \{1,\ldots,s\}$ be given. Assume that a row $x \in X$ possesses v as a simple witness (i.e., $v_i \in x_i$ for all $i \in I$). We set $y_i = x_i$ whenever $i \notin I$, and otherwise

$$y_i = \begin{cases} x_i & \text{if } \{u_i,v_i\} \subseteq x_i, \\ x_i - \{v_i\} \cup \{u_i\} & \text{otherwise.} \end{cases}$$

The function f, with $f(x) = (y_1,\ldots,y_s)$, is an injection mapping elements of X with witness v to elements of X with witness u. The claim follows. □

The next lemma exhibits the crucial advantage of monotone sets, namely, that a large monotone set X possesses a witness w_X (with a large core set I) such that w_X

- has only large components and
- is nevertheless still quite likely.

In particular we describe the relationship between the size of X (expressed by α) on one side and the component size of w_X (expressed by β), the probability of w_X (expressed by ε), and the size of the core (expressed by δ) on the other side.

LEMMA 6.5. *$\alpha \in [0,1]$ and $\beta,\delta,\varepsilon \in [0,1/2]$ are nonnegative real numbers with*

$$\alpha > 4^{1/n} \cdot e^{-\varepsilon^2 \cdot \beta \cdot \delta/4} \quad \text{and} \quad \varepsilon \geq \frac{n}{\beta \cdot m}. \tag{6.3}$$

Let $X \subseteq \mathcal{P}^{(s)}_{m,n}$ be monotone decreasing with $|X| = \alpha^{n \cdot s} \cdot \binom{m}{n}^s$.

(a) *With probability at least $1-2^{-s}$, an element $(x_1,\ldots,x_s) \in X$ has the property that*

$$|\{(1-\beta)\cdot m+1,\ldots,m\} \cap x_i| \geq (1-\varepsilon)^2 \cdot \beta \cdot n$$

for at least $(1-\delta)\cdot s$ components i.

(b) *There is a subset* $I \subseteq \{1,\dots,s\}$ *with* $|I| = (1-\delta)\cdot s$ *such that for any subset* $J \subseteq I$ *of size* k *the simple witness* w^J*, with core* J *and* $w^J_j = (1-\beta)\cdot m$ *for* $j \in J$*, satisfies*

$$p_X(w^J) \geq \frac{(1-\varepsilon)^{2k}}{2\cdot\binom{s}{\delta\cdot s}}\cdot\left(\frac{n}{m}\right)^k.$$

Proof. (a) We first determine an upper bound for the probability

$$p_E = \mathrm{prob}[|y \cap \{(1-\beta)\cdot m+1,\dots,m\}| \leq E],$$

where y is a random subset of $\{1,\dots,m\}$ of size n. We claim that

$$p_E \leq \mathrm{prob}\left[\sum_{i=1}^{n} Y_i \leq E\right], \tag{6.4}$$

where

$$Y_i = \begin{cases} 1 & \text{with probability } \beta - \frac{n}{m}, \\ 0 & \text{with probability } 1-\beta+\frac{n}{m}. \end{cases}$$

Assume that we pick n elements from $\{1,\dots,m\}$ without replacement and call the selection of an element from $\{(1-\beta)\cdot m+1,\dots,m\}$ a success. Then the success probability in the $n+1$st attempt will be minimal if all previous attempts have been successful, and hence the success probability is always at least $\frac{\beta\cdot m-n}{m} = \beta - \frac{n}{m}$. Hence (6.4) follows, since the probability of staying below the threshold E is higher when we assume a smaller success probability.

We have $\varepsilon \geq \frac{n}{\beta\cdot m}$ (with (6.3)) and therefore $1-\frac{n}{\beta\cdot m} \geq 1-\varepsilon$. Thus,

$$\beta - \frac{n}{m} \geq (1-\varepsilon)\cdot\beta \ \text{ and } \ \beta-\frac{n}{m} \geq \frac{\beta}{2} \ \left(\text{since } \varepsilon \leq \frac{1}{2}\right). \tag{6.5}$$

Observe that $E^* = (\beta - \frac{n}{m})\cdot n$ is the expected value of $\sum_{i=1}^{n} Y_i$. We set $E = (1-\varepsilon)\cdot E^*$ and obtain as a consequence of Chernoff's bound

$$\begin{aligned} p_E &\leq \mathrm{prob}\left[\sum_{i=1}^{n} Y_i \leq (1-\varepsilon)\cdot E^*\right] \\ &\leq e^{-\varepsilon^2\cdot E^*/2}. \end{aligned}$$

Let J be a subset of $\{1,\dots,s\}$, and let $p_E(J)$ be the probability that a randomly chosen element $x = (x_1,\dots,x_s)$ of $\mathcal{P}^{(s)}_{m,n}$ has the property that x_j, for all $j \in J$, intersects $\{(1-\beta)\cdot m+1,\dots,m\}$ in at most E elements. Then we get

$$p_E(J) \leq e^{-\varepsilon^2\cdot E^*\cdot|J|/2}$$

and, as a consequence,

$$\sum_{J\subseteq\{1,\dots,s\},|J|=\delta\cdot s} p_E(J) \leq 2^s\cdot e^{-\varepsilon^2\cdot E^*\cdot\delta\cdot s/2}.$$

In other words, the probability that a randomly chosen element x intersects $\{(1-\beta)\cdot m+1,\ldots,m\}$ in at most E elements for $\delta\cdot s$ components is small. On the other hand, $|X|=\alpha^{s\cdot n}\cdot\binom{m}{n}^s$ with

$$\begin{aligned}\alpha^{s\cdot n} &\geq 2^{2s}\cdot e^{-\varepsilon^2\cdot\beta\cdot\delta\cdot s\cdot n/4} \qquad \text{because of (6.3)}\\ &\geq 2^{2s}\cdot e^{-\varepsilon^2\cdot(\beta-\frac{n}{m})\cdot n\cdot\delta\cdot s/2} \qquad \text{because of (6.5)}\\ &= 2^s\cdot 2^s\cdot e^{-\varepsilon^2\cdot E^*\cdot\delta\cdot s/2},\end{aligned}$$

and hence

$$\sum_{J\subseteq\{1,\ldots,s\},|J|=\delta\cdot s} p_E(J)\leq 2^{-s}\cdot\frac{|X|}{\binom{m}{n}^s}.$$

Therefore all but $|X|/2^s$ elements $(x_1,\ldots,x_s)$ of X have the property that $|\{(1-\beta)\cdot m+1,\ldots,m\}\cap x_i|>E$ for at least $(1-\delta)\cdot s$ components i. The claim follows, since $E=(1-\varepsilon)\cdot E^*=(1-\varepsilon)\cdot(\beta-\frac{n}{m})\cdot n\geq(1-\varepsilon)^2\cdot\beta\cdot n$ (with (6.5)).

(b) According to (a) there is a set $I\subseteq\{1,\ldots,s\}$ with $|I|=(1-\delta)\cdot s$ and a subset $X_0\subseteq X$ such that

- $|X_0|\geq\frac{|X|}{2\cdot\binom{s}{\delta\cdot s}}$ and
- for all $i\in I$ and for all $x\in X_0$, $|\{(1-\beta)\cdot m+1,\ldots,m\}\cap x_i|>E$.

Hence for every subset $J\subseteq I$ with $|J|=k$,

$$\begin{aligned}&\sum_{u\in\text{Witness}_{J,1},u_j>(1-\beta)\cdot m\text{ for all }j\in J} p_X(u)\\ &\quad=\sum_{u\in\text{Witness}_{J,1},u_j>(1-\beta)\cdot m\text{ for all }j\in J}\frac{|\{x\in X\ :\ u\in\text{Witness}_{J,1}(x)\}|}{|X|}\\ &\quad=\sum_{x\in X}\frac{|\{u\in\text{Witness}_{J,1}(x)\ :\ u_j>(1-\beta)\cdot m\text{ for all }j\in J\}|}{|X|}\\ &\quad=\sum_{x\in X}\frac{\prod_{j\in J}|\{(1-\beta)\cdot m+1,\ldots,m\}\cap x_j|}{|X|}.\end{aligned}$$

We can now bring X_0 into play and get

$$\sum_{x\in X}\frac{\prod_{j\in J}|\{(1-\beta)\cdot m+1,\ldots,m\}\cap x_j|}{|X|}\geq\sum_{x\in X_0}\frac{E^k}{|X|}\geq\frac{|X|}{2\cdot\binom{s}{\delta\cdot s}}\cdot\frac{E^k}{|X|}=\frac{E^k}{2\cdot\binom{s}{\delta\cdot s}}.$$

As a consequence of Proposition 6.4, the witness w^J (with core J and $w^J_j=(1-\beta)\cdot m$ for all $j\in J$) has the highest probability of all simple witnesses u with core J and $u_j>(1-\beta)\cdot m$ for all $j\in J$ and hence

$$\begin{aligned}p_X(w^J)&\geq\frac{E^k}{2\cdot\binom{s}{\delta\cdot s}\cdot(\beta\cdot m)^k}\\ &\geq\frac{(1-\varepsilon)^k\cdot(\beta-\frac{n}{m})^k n^k}{2\cdot\binom{s}{\delta\cdot s}\cdot(\beta\cdot m)^k}\cdot\\ &\geq\frac{(1-\varepsilon)^k}{2\cdot\binom{s}{\delta\cdot s}}\cdot\frac{(\beta-\frac{n}{m})^k}{\beta^k}\cdot\left(\frac{n}{m}\right)^k\\ &\geq\frac{(1-\varepsilon)^{2k}}{2\cdot\binom{s}{\delta\cdot s}}\cdot\left(\frac{n}{m}\right)^k \qquad \text{because of (6.5)},\end{aligned}$$

and this was to be shown. □

The results of this section are crucial for proving Lemma 5.1. This will be done in the next section.

7. Large submatrices and simple witnesses. As a first step in the proof of Lemma 5.1, we show that witnesses from a relatively small set W of witnesses will constitute a correspondingly small minority among simple witnesses for the rows of a relatively large matrix. The next lemma makes the size dependencies explicit and will be used to show that small sets of forbidden witnesses have only little impact.

LEMMA 7.1. *$\alpha \in [0,1]$ and $\varepsilon \in [0,1/6]$ are nonnegative real numbers with*

$$\alpha > 4^{1/n} \cdot e^{-\varepsilon^3 \cdot (1-\varepsilon)/4} \quad \text{and} \quad \varepsilon \geq \frac{n}{(1-\varepsilon)\cdot m}.$$

Assume that X is a subset of $\mathcal{P}^{(s)}_{m,n}$ of size $\alpha^{s\cdot n} \cdot \binom{m}{n}^s$. For a subset $I \subseteq \{1,\ldots,s\}$ of size $k \geq s/2$ let W be a subset of $\mathrm{Witness}_{I,1}$ *with $|W| \leq (\varepsilon \cdot m)^{(1-11\varepsilon)\cdot k}$. Then*

$$\sum_{w \in W} p_X(w) \leq 2 \cdot e^{-3\varepsilon \cdot k} \cdot n^k.$$

Proof. We first show, using the methods in the proof of Lemma 6.3, that one can assume that W is monotone decreasing, i.e., if $u \leq v$ componentwise and $v \in W$, then also $u \in W$. If W is not monotone decreasing, then there will be an element $w \in W$ such that, without loss of generality, $w_1 = \{\xi\}$ and $w' = (\{\xi - 1\}, w_2, \ldots, w_s) \notin W$.

For $x \in X$ set $x' = (x_1 - \{\xi\} \cup \{\xi - 1\}, x_2, \ldots, x_s)$ and define

$$\mathrm{decrease}_\xi(x) = \begin{cases} x' & \text{if } x_1 \cap \{\xi - 1, \xi\} = \{\xi\} \text{ and } x' \notin X, \\ x & \text{otherwise.} \end{cases}$$

Analogously define the function $\mathrm{decrease}_\xi(u)$ for $u \in W$. We replace X and W by

$$X' = \{\mathrm{decrease}_\xi(x) \mid x \in X\} \text{ and } W' = \{\mathrm{decrease}_\xi(u) \in W \mid u \in W\},$$

and observe first that $|X| = |X'|$ and $|W| = |W'|$. Finally we claim that

$$\sum_{w \in W} p_X(w) \leq \sum_{w' \in W'} p_{X'}(w').$$

Let u be an element of W. If $\{\xi - 1, \xi\} \cap u_1 = \emptyset$, then $u \in W'$ and $p_X(u) = p_{X'}(u)$. Otherwise u has a twin u', i.e., $u \neq u'$ and one of u and u' results from the other by an application of decrease. If $u' \in W$, then both belong to W' and $p_X(u) + p_X(u') = p_{X'}(u) + p_{X'}(u')$. Otherwise $u' \in W' - W$ and therefore $u'_1 = \{\xi - 1\}$. Hence $p_X(u) \leq p_{X'}(u')$.

Thus, after repeating the above procedure suitably often, we obtain a subset $X^* \subseteq \mathcal{P}^{(s)}_{m,n}$ (with $|X^*| = |X|$) and a monotone decreasing subset W^* (with $|W^*| = |W|$), such that

$$\sum_{w \in W} p_X(w) \leq \sum_{w^* \in W^*} p_{X^*}(w^*).$$

Let I be a subset of $\{1,\ldots,s\}$ of size $k \geq s/2$. Since $|W^*| = |W|$ and $|W| \leq (\varepsilon \cdot m)^{(1-11\varepsilon)\cdot k}$ and since W^* is monotone decreasing, *each* element of W^* has at least $11\varepsilon \cdot k$ components of value at most $\varepsilon \cdot m$, since otherwise

$$|W^*| > (\varepsilon \cdot m)^{(1-11\varepsilon)\cdot k}.$$

Thus all elements of W^* have a large number of small components. We say that a component i is *regular* for $(x_1, \ldots, x_s)$ provided

$$|\{\varepsilon \cdot m + 1, \ldots, m\} \cap x_i| \geq (1-\varepsilon)^3 \cdot n.$$

Finally we call x *conventional* if x has at least $(1-\varepsilon) \cdot s$ regular components. We apply Lemma 6.5(a) (with $\beta = 1-\varepsilon$ and $\delta = \varepsilon$) and obtain, with probability at least $1 - 2^{-s}$, that an element $x^* \in X^*$ is conventional. Fix an arbitrary conventional element x^* and distinguish exactly $(1-\varepsilon) \cdot s$ of its regular components. We claim that

$$|\{w \in W^* \mid w \in \text{Witness}_{I,1}(x^*)\}| \leq e^{-3\varepsilon \cdot k} \cdot n^k. \tag{7.1}$$

Let E be the expected number of small components (i.e., of value at most $\varepsilon \cdot m$) of a witness $w \in \text{Witness}_{I,1}(x^*)$, where we consider only the distinguished regular components of x^*. Then $E \leq (k - \varepsilon \cdot s) \cdot (1 - (1-\varepsilon)^3) \leq (k - \varepsilon \cdot s) \cdot 3\varepsilon \leq 3\varepsilon \cdot k$. We choose β, with $\beta \cdot E = 9\varepsilon \cdot k$, and obtain

$$\text{prob}[w \text{ has at least } 9\varepsilon \cdot k \text{ small components} \mid w \in \text{Witness}_{I,1}(x^*)] \leq e^{-E \cdot (\beta-1)^2/3}$$

with the Chernoff bound. We have $E \cdot (\beta - 1) = E \cdot \beta - E \geq 6\varepsilon \cdot k$ and hence $\beta - 1 \geq 2$ as well as $E \cdot (\beta-1)^2 \geq 12\varepsilon \cdot k$. Thus the probability of having at least $9\varepsilon \cdot k$ small components is bounded by $e^{-12\varepsilon \cdot k/3} \leq e^{-3\varepsilon \cdot k}$.

If we also consider the at most $\varepsilon \cdot s$ irregular components of I, we obtain, with probability at most $e^{-3\varepsilon \cdot k}$, at least $9\varepsilon \cdot k + \varepsilon \cdot s \leq 11\varepsilon \cdot k$ components with value at most $\varepsilon \cdot m$. Claim (7.1) follows, since elements of W^* have at least $11\varepsilon \cdot k$ components with value at most $\varepsilon \cdot m$.

We conclude by also considering (the few) unconventional elements of X^* and obtain

$$\begin{aligned}
\sum_{w \in W^*} p_{X^*}(w) &= \sum_{x \in X^*} \frac{|\{w \in W^* \mid w \in \text{Witness}_{I,1}(x^*)\}|}{|X^*|} \\
&\leq \frac{|X^*|}{2^s} \cdot \frac{n^k}{|X^*|} + \sum_{x \in X^*,\ x \text{ is conventional}} e^{-3\varepsilon \cdot k} \cdot \frac{n^k}{|X^*|} \\
&\leq \frac{n^k}{2^s} + e^{-3\varepsilon \cdot k} \cdot n^k = \left(\frac{1}{2^s} + e^{-3\varepsilon \cdot k}\right) \cdot n^k \leq 2 \cdot e^{-3\varepsilon \cdot k} \cdot n^k,
\end{aligned}$$

where the last inequality follows, since $\varepsilon \leq \frac{1}{4}$. □

Proof of Lemma 5.1. We have to show that a large subset $I \subseteq \{1, \ldots, s\}$ (of size k) exists, such that many quite likely (simple) witnesses w (with core I) are common to X and Y even if witnesses in a small set W are not counted. We may assume that X and Y are large ($|X| = \alpha_1^{s \cdot n} \cdot \binom{m}{n}^s$ and $|Y| = \alpha_2^{s \cdot n} \cdot \binom{m}{n}^s$) and that W is small ($|W| \leq (\varepsilon_2 \cdot m)^{(1-11 \cdot \varepsilon_2) \cdot s}$).

We start by making the transition to monotone sets X' and Y' according to Lemma 6.3 (with $|X| = |X'|$ and $|Y| = |Y'|$). Next, Lemma 6.5, applied for $\varepsilon = 2\beta = \delta = \varepsilon_1$, provides a subset $I_{X'} \subseteq \{1, \ldots, s\}$ of size $(1-\varepsilon_1) \cdot s$ such that

$$p_{X'}(u^J) \geq \frac{(1-\varepsilon_1)^{2l}}{2 \cdot \binom{s}{\varepsilon_1 \cdot s}} \cdot \left(\frac{n}{m}\right)^l$$

for any subset $J \subseteq I_{X'}$ of size l, where $u_j^J = (1-\varepsilon_1/2)\cdot m$ for all $j \in J$. The situation for Y' is analogous, and we obtain a subset $I_{Y'} \subseteq \{1,\ldots,s\}$ of size $(1-\varepsilon_1)\cdot s$ such that

$$p_{Y'}(v^J) \geq \frac{(1-\varepsilon_1)^{2l}}{2\cdot\binom{s}{\varepsilon_1\cdot s}}\cdot\left(\frac{n}{m}\right)^l$$

for any subset $J \subseteq I_{Y'}$ of size l, where $v_j^J = (1-\varepsilon_1/2)\cdot m$ for all $j \in J$. We set $I = I_{X'} \cap I_{Y'}$ and obviously $k := |I| \geq (1-2\cdot\varepsilon_1)\cdot s$.

We go back to Lemma 6.3 and apply it for the system (X, f, Y, g, I), where

$$f(x) = W \cap \mathrm{Witness}_{I,1}(x),$$
$$g(y) = W \cap \mathrm{Witness}_{I,1}(y).$$

With Lemma 6.3 we obtain functions $f' : X' \to \mathcal{P}(\mathrm{Witness}_{I,1})$ and $g' : Y' \to \mathcal{P}(\mathrm{Witness}_{I,1})$ defining the new sets of forbidden witnesses such that

$$\sum_{x\in X}|f(x)| = \sum_{x'\in X'}|f'(x')| \text{ and } \sum_{y\in Y}|g(y)| = \sum_{y'\in Y'}|g'(y')|. \tag{7.2}$$

Finally we set

$$p_X^*(u) = |\{x \in X \ : \ u \in \mathrm{Witness}_{I,1}(x) - f(x)\ \}|/|X|,$$
$$p_Y^*(u) = |\{y \in Y \ : \ u \in \mathrm{Witness}_{I,1}(y) - g(y)\ \}|/|Y|$$

and introduce $p_{X'}^*(u), p_{Y'}^*(u)$ analogously. Then we get

$$\sum_{w\in\mathrm{Witness}_{I,1}} p_X^*(w)\cdot p_Y^*(w) \geq \sum_{w\in\mathrm{Witness}_{I,1}} p_{X'}^*(w)\cdot p_{Y'}^*(w)$$

as a consequence of Remark 6.1. We apply Lemma 7.1 (with $\varepsilon = \varepsilon_2$) and utilize (7.2) to get

$$\sum_{x'\in X'}|f'(x')| = \sum_{x\in X}|f(x)| = |X|\cdot\sum_{w\in W}p_X(w) \leq 2|X|\cdot e^{-3\varepsilon_2\cdot k}\cdot n^k \text{ and}$$
$$\sum_{y'\in Y'}|g'(y')| = \sum_{y\in Y}|f(y)| = |Y|\cdot\sum_{w\in W}p_Y(w,) \leq 2|Y|\cdot e^{-3\varepsilon_2\cdot k}\cdot n^k.$$

Call a simple witness $u \in \mathrm{Witness}_{I,1}$ *crucial* if $\varepsilon_1\cdot m/2+1 \leq u_i \leq (1-\varepsilon_1/2)\cdot m$ for all $i \in I$. Observe that there are $(1-\varepsilon_1)^k\cdot m^k$ crucial witnesses. Let

$$U = \{u \ : \ u \in \mathrm{Witness}_{I,1} \text{ is crucial and } p_{X'}^*(u) < p_{X'}(u)/2\}.$$

We claim that $|U| \leq \frac{1}{4}\cdot(1-\varepsilon_1)^k\cdot m^k$. The difference between $p_{X'}^*$ and $p_{X'}$ is due to forbidden witnesses and, since

$$\sum_{u\in U}|\{x' \in X' \ : \ u \in \mathrm{Witness}_{I,1}(x') - f'(x')\ \}|$$
$$\geq \sum_{u\in U}|\{x' \in X' \ : \ u \in \mathrm{Witness}_{I,1}(x')\ \}| - \sum_{x'\in X'}|f'(x')|,$$

we obtain

$$\begin{aligned}|X'| \cdot \sum_{u \in U} p^*_{X'}(u) &\geq |X'| \cdot \sum_{u \in U} p_{X'}(u) - \sum_{x' \in X'} |f'(x')| \\ &\geq |X'| \cdot \sum_{u \in U} p_{X'}(u) - 2|X| \cdot e^{-3\varepsilon_2 \cdot k} \cdot n^k.\end{aligned}$$

As a consequence, since $|X| = |X'|$,

$$\sum_{u \in U} p^*_{X'}(u) \geq \sum_{u \in U} p_{X'}(u) - 2 \cdot e^{-3\varepsilon_2 \cdot k} \cdot n^k. \tag{7.3}$$

Let u be a crucial witness. Then with Proposition 6.4 and Lemma 6.5,

$$p_{X'}(u) \geq p_{X'}(u^I) \geq \frac{(1-\varepsilon_1)^{2k}}{2 \cdot \binom{s}{\varepsilon_1 \cdot s}} \cdot \left(\frac{n}{m}\right)^k,$$

and therefore, assuming $|U| > \frac{1}{4} \cdot (1-\varepsilon_1)^k \cdot m^k$,

$$\begin{aligned}\sum_{u \in U} p_{X'}(u) &\geq |U| \cdot \frac{(1-\varepsilon_1)^{2k}}{2 \cdot \binom{s}{\varepsilon_1 \cdot s}} \cdot \left(\frac{n}{m}\right)^k = \frac{|U|}{m^k} \cdot \frac{(1-\varepsilon_1)^{2k}}{2 \cdot \binom{s}{\varepsilon_1 \cdot s}} \cdot n^k \\ &> \frac{1}{4} \cdot (1-\varepsilon_1)^k \cdot \frac{(1-\varepsilon_1)^{2k}}{2 \cdot \binom{s}{\varepsilon_1 \cdot s}} \cdot n^k = \frac{1}{8} \cdot \frac{(1-\varepsilon_1)^{3k}}{\binom{s}{\varepsilon_1 \cdot s}} \cdot n^k \\ &\geq 4 \cdot e^{-3\varepsilon_2 \cdot k} \cdot n^k \qquad \text{because of (5.7).}\end{aligned}$$

Thus, if $|U| > \frac{1}{4} \cdot (1-\varepsilon_1)^k \cdot m^k$, then with (7.3),

$$\sum_{u \in U} p^*_{X'}(u) \geq \sum_{u \in U} p_{X'}(u) - 2 \cdot e^{-3\varepsilon_2 \cdot k} \cdot n^k \geq \sum_{u \in U} p_{X'}(u)/2,$$

and $p^*_{X'}(u) \geq p_{X'}(u)/2$ for at least one witness $u \in U$. This contradicts the definition of U, and therefore $|U| \leq \frac{1}{4} \cdot (1-\varepsilon_1)^k \cdot m^k$.

We carry out the same argument for the set Y of columns and obtain that for at least one half of all crucial witnesses u,

$$p^*_{X'}(u) \geq \frac{p_{X'}(u)}{2} \geq \frac{(1-\varepsilon_1)^{2k}}{4 \cdot \binom{s}{\varepsilon_1 \cdot s}} \cdot \left(\frac{n}{m}\right)^k \quad \text{and}$$

$$p^*_{Y'}(u) \geq \frac{p_{Y'}(u)}{2} \geq \frac{(1-\varepsilon_1)^{2k}}{4 \cdot \binom{s}{\varepsilon_1 \cdot s}} \cdot \left(\frac{n}{m}\right)^k.$$

Since

$$\begin{aligned}\sum_{u \in \text{Witness}_{I,1} - W} p_X(u) \cdot p_Y(u) &= \sum_{u \in \text{Witness}_{I,1}} p^*_X(u) \cdot p^*_Y(u) \\ &\geq \sum_{u \in \text{Witness}_{I,1}} p^*_{X'}(u) \cdot p^*_{Y'}(u) \\ &\geq \sum_{u \text{ is crucial}} p^*_{X'}(u) \cdot p^*_{Y'}(u) \\ &\geq \frac{1}{2} \cdot (1-\varepsilon_1)^k \cdot m^k \cdot \frac{(1-\varepsilon_1)^{4k}}{16 \cdot \binom{s}{\varepsilon_1 \cdot s}^2} \cdot \left(\frac{n}{m}\right)^{2k} \\ &\geq \frac{1}{32} \cdot \frac{(1-\varepsilon_1)^{5k}}{\binom{s}{\varepsilon_1 \cdot s}^2} \cdot \left(\frac{n^2}{m}\right)^k,\end{aligned}$$

the claim follows. □

8. Conclusion. We have shown that a reduction in the number of advice bits by a logarithmic factor may result in an almost optimal increase for nondeterministic communication. This result was obtained for the language $D^{(s)}_{m,n}$, a direct sum version of the disjointness language, by

1. first showing that any 1-chromatic submatrix of the communication matrix has few *simple* witnesses testifying for most entries and
2. then utilizing this property by showing that a large 1-chromatic submatrix has a large number of quite likely *double* witnesses.

A deterministic protocol partitions the communication matrix into submatrices, and hence the number of double witnesses is at least as large as the sum, over all 1-chromatic submatrices M, of the number of double witnesses of M. Thus a deterministic protocol that exchanges few bits and accepts many inputs cannot be correct, since it generates too many double witnesses. Step 2 is verified by showing that monotone submatrices will have the smallest number of entries with a common simple witness and hence follows a qualitative approach.

Consequently we obtained a family $L_n \subseteq \{0,1\}^{2n}$ of languages with

$$\mathrm{nc}_{o(\sqrt{n}/\log_2 n)}(L_n) = \Omega\left(\frac{\mathrm{svnc}(L_n)^2}{\log_2 n}\right) = \Omega\left(\frac{n}{\log_2 n}\right), \tag{8.1}$$

and hence an almost quadratic gap between self-verifying nondeterministic communication and advice-restricted nondeterminism. As an immediate consequence of this gap we also obtained an almost quadratic gap between self-verifying nondeterministic communication and Monte-Carlo communication, namely,

$$\mathrm{mcc}(L_n) \geq \mathrm{nc}_{o(\sqrt{n}/\log_2 n)}(L_n) = \Omega\left(\frac{\mathrm{svnc}(L_n)^2}{\log_2 n}\right) = \Omega\left(\frac{n}{\log_2 n}\right).$$

We close with two questions. Can the logarithmic gap in (8.1) be removed? Are there further application areas for the new lower bound method?

Acknowledgment. Paul Beame observed that Theorem 1.1 separates Monte-Carlo communication from self-verifying nondeterministic communication.

REFERENCES

[1] H. Abelson, *Lower bound on an information transfer in distributed computation*, in Proceedings of the 19th IEEE Symposium on Foundations of Computer Science, Ann Arbor, MI, 1978, pp. 151–158.

[2] A. V. Aho, J. D. Ullman, and M. Yannakakis, *On notions of information transfer in VLSI circuits*, in Proceedings of the 15th Annual ACM Symposium on Theory of Computing, Boston, 1983, pp. 133–139.

[3] L. Babai, P. Frankl, and J. Simon, *Complexity classes in communication complexity theory*, in Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, Toronto, 1986, pp. 337–347.

[4] L. Babai, N. Nisan, and M. Szegedy, *Multiparty protocols and logspace-hard pseudorandom sequences*, J. Comput. System Sci., 45 (1992), pp. 204–232.

[5] P. Beame and J. Lawry, *Randomized versus nondeterministisc communication complexity*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, Victoria, BC, 1992, pp. 188–199.

[6] R. Canetti and O. Goldreich, *Bounds on tradeoffs between randomness and communication complexity*, Comput. Complexity, 3 (1993), pp. 141–167.

[7] M. Fleischer, H. Jung, and K. Mehlhorn, *A communication-randomness tradeoff for two-processor systems*, Inform. and Comput., 116 (1995), pp. 155–161.

[8] P. Frankl and Z. Füredi, *A short proof for a theorem of Harper about Hamming spheres*, Discrete Math., 34 (1981), pp. 311–313.

[9] J. Hromkovič, *Communication Complexity and Parallel Computing*, EATCS Texts in Theoretical Computer Science, Springer-Verlag, Berlin, 1997.

[10] J. Hromkovič and G. Schnitger, *Nondeterministic communication with a limited number of advice bits*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, Philadelphia, 1996, pp. 551–560.

[11] J. Ja'Ja', V. K. Prasanna Kumar, and J. Simon, *Information transfer under different sets of protocols*, SIAM J. Comput., 13 (1984), pp. 840–849.

[12] T. S. Jayram, R. Kumar, and D. Sivakumar, *Two applications of information complexity*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, 2003, pp. 673–682.

[13] H. Klauck, *Lower bounds for computation with limited nondeterminism*, in Proceedings of the 13th Annual IEEE Conference on Computational Complexity, Buffalo, NY, 1998, pp. 141–152.

[14] B. Kalyanasundaram and G. Schnitger, *Communication complexity and lower bounds for sequential machines*, in Festschrift zum 60. Geburtstag von Günter Hotz, Teubner Verlag, Stuttgart, 1992, pp. 253–268.

[15] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.

[16] T. Lengauer, *VLSI theory*, in Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity, J. van Leeuwen, ed., Elsevier, New York, 1990, pp. 835–868.

[17] K. Mehlhorn and E. Schmidt, *Las Vegas is better than determinism in VLSI and distributed computing*, in Proceedings of the 14th Annual ACM Symposium on Theory of Computing, San Francisco, 1982, pp. 330–337.

[18] I. Newman, *Private versus common random bits in communication complexity*, Inform. Process. Lett., 39 (1991), pp. 67–71.

[19] R. Raz and A. Wigderson, *Monotone circuits for matching require linear depth*, J. ACM, 39 (1992), pp. 736–744.

[20] M. Yannakakis, *Expressing combinatorial optimization problems by linear programs*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, Chicago, 1988, pp. 223–228.

[21] A. C. Yao, *Some complexity questions related to distributive computing*, in Proceedings of the 11th Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1979, pp. 209–213.

SIAM J. COMPUT.
Vol. 33, No. 1, pp. 69–93

QUALITY MESHING WITH WEIGHTED DELAUNAY REFINEMENT*

SIU-WING CHENG† AND TAMAL K. DEY‡

Abstract. Delaunay meshes with bounded circumradius to shortest edge length ratio have been proposed in the past for quality meshing. The only poor quality tetrahedra, called *slivers*, that can occur in such a mesh can be eliminated by the *sliver exudation* method. This method has been shown to work for periodic point sets, but not with boundaries. Recently a randomized point-placement strategy has been proposed to remove slivers while conforming to a given boundary. In this paper we present a deterministic algorithm for generating a weighted Delaunay mesh which respects the input boundary and has no poor quality tetrahedron including slivers. As in previous work, we assume that no input angle is acute. Our result is achieved by combining the weight pumping method for sliver exudation and the Delaunay refinement method for boundary conformation.

Key words. tetrahedral mesh generation, mesh quality, algorithms, computational geometry, Delaunay triangulation, weighted Delaunay refinement, sliver

AMS subject classifications. 52B70, 52C99, 68U05, 68U07

DOI. 10.1137/S0097539703418808

1. Introduction.

In finite element methods a three-dimensional domain is often partitioned with tetrahedra. The quality of their shapes influences the quality of the finite element solution [21]. This motivated the decade-long research on generating meshes with guaranteed aspect ratio called *quality meshes* [1, 3, 5, 6, 7, 18, 19, 20]. A considerable literature has built up on the subject; see the surveys and books [2, 11, 14, 22]. We review only a few of them in the context of the work in this paper.

Bern, Eppstein, and Gilbert [3] pioneered a quadtree-based triangulation approach for producing quality meshes with close to optimal size in two dimensions. Mitchell and Vavasis [18] extended this technique to triangulate polyhedra with guaranteed aspect ratio in higher dimensions. This line of work provided many crucial insights into the problem, though the elements produced by this method have a biased alignment because of the axis-parallel boxes used in quadtree/octtree subdivisions. Delaunay-based triangulations do not have this problem, and they are widely used in mesh generation for their uniqueness and many other nice properties; see [14, 11]. As a result, researchers have also concentrated on computing meshes as a subcomplex of a Delaunay mesh with guaranteed quality. Chew proposed a simple circumcenter insertion method for the problem in two dimensions [5] which produces a uniform mesh of quality triangles. Ruppert [19], in a pioneering work called Delaunay refinement, showed how circumcenter insertion can be used to produce a quality graded mesh with optimal size.

Many of the concepts, including the analysis of the *local feature size*, introduced by Ruppert are the basis of the further developments in this area. Shewchuk made an important contribution in extending the Delaunay refinement to three-dimensional

*Received by the editors March 14, 2003; accepted for publication September 11, 2003; published electronically November 14, 2003. This research was supported by the Research Grant Council, Hong Kong, China, project HKUST6088/99E, and by the NSF under grant CCR-9988216.
http://www.siam.org/journals/sicomp/33-1/41880.html
†Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong (scheng@cs.ust.hk).
‡Department of Computer and Information Sciences, The Ohio State University, Columbus, OH 43210 (tamaldey@cis.ohio-state.edu).

domains with boundaries [20]. This refinement eliminates any tetrahedron that has a large ratio of its circumradius to the shortest edge length. Consequently the resulting mesh satisfies the *radius-edge ratio property*; i.e., all tetrahedra have radius-edge ratios below a threshold. Although most of the poor quality tetrahedra are removed by the radius-edge ratio property, one type of bad tetrahedra called *slivers* are not eliminated; see [8]. Slivers are formed by four points placed almost uniformly around the equator of a sphere. Slivers have bounded radius-edge ratio, but they have negligible volume, which makes them poor quality. Cheng et al. [7] proposed the *sliver exudation* method to get rid of the slivers from a Delaunay mesh that already have the radius-edge ratio property. They introduced the *pumping* technique, which assigns a weight to each vertex, which prohibits any sliver to be incident on them in the weighted Delaunay triangulation. Subsequently, Edelsbrunner et al. [12] developed a method for perturbing the points so that the unweighted Delaunay tetrahedralization has the radius-edge ratio property and is sliver-free.

Unfortunately, the algorithms of [7, 12] could not handle boundaries and were applied to periodic point sets. In a recent work, Edelsbrunner and Guoy [13] experimented with the sliver exudation method, which reveals that the technique is very effective in eliminating slivers. After sliver exudation, almost all tetrahedra have angles greater than 5° (except that some tetrahedra with angles less than 5° survive near the boundary). Thus, boundary handling remains a challenge. Recently, Li and Teng [15] showed that it is possible to construct a sliver-free Delaunay mesh, in the presence of boundaries, with a randomized point-placement strategy extended from that of Chew [6]. A sliver is destroyed by inserting a random point near its circumcenter. The analysis of this algorithm is a nice example of the confluence of the results developed over the years by Chew [6], Ruppert [19], Shewchuk [20], and Cheng et al. [7].

In this paper we present the first *deterministic* algorithm to construct a Delaunay mesh, with the radius-edge ratio property and without any sliver, of a three-dimensional domain with boundaries. We combine Delaunay refinement with sliver exudation and obtain what we call the *weighted Delaunay refinement*. We show that this technique produces a graded mesh with asymptotically optimal size. Our work can be viewed as an advance along the line of research initiated by Chew and Ruppert [5, 19] and carried forward by Dey, Bajaj, and Sugihara [8], Shewchuk [20], Cheng et al. [7], Edelsbrunner et al. [12], and Li and Teng [15]. Encouraged by the experimental results of Edelsbrunner and Guoy [13], we believe that weighted Delaunay refinement is eminently practical. Their experiments show that, after sliver exudation, relatively few slivers near the domain boundary survive. Thus, we expect that most slivers can be destroyed without adding extra points.

The rest of the paper is organized as follows. Section 2 describes the basic definitions. Section 3 presents our algorithm, and its behavior is analyzed in sections 4, 5, and 6. Section 7 proves the guarantees achieved by our algorithm. We conclude in section 8.

2. Definitions. We need the following definitions, most of which have been introduced in earlier works.

Quality of tetrahedra. The volumes of tetrahedra in a normalized sense capture their quality. Poor quality tetrahedra have small normalized volume. Let R, L, and V be the circumradius, shortest edge length, and volume, respectively, of a tetrahedron τ. We characterize τ by two ratios $\rho(\tau) = R/L$ and $\sigma(\tau) = V/L^3$. If $\rho(\tau)$ exceeds a threshold ρ_0, then we call it *skinny*. If $\rho(\tau) \leq \rho_0$ and $\sigma(\tau)$ does not exceed a threshold

σ_0, then we call τ a sliver. As in previous work [7, 15], we will show that there exist ρ_0 and σ_0 independent of the domain such that our algorithm does not produce any skinny tetrahedron or sliver with respect to ρ_0 and σ_0.

Piecewise linear complex. The domain to be meshed is a bounded volume, and its boundary is presented using a *piecewise linear complex* (PLC). A collection $\mathcal{P}$ of vertices, segments, and facets in $\mathbb{R}^3$ is called a PLC (i) if all elements on the boundary of an element in $\mathcal{P}$ also belong to $\mathcal{P}$ and (ii) if any two elements intersect, their intersection is a lower-dimensional element in $\mathcal{P}$.

Input. We assume that the domain to be meshed is a convex bounded volume containing a collection of vertices, segments, and facets represented as a PLC. An input angle is the angle between two segments sharing a vertex, a segment and a facet sharing one vertex, or two facets sharing a vertex or a segment. We assume that no input angle is acute. If the input PLC does not bound a convex volume, we enclose it in a large box, mesh the inside of the box, and keep only the tetrahedra covering the original domain. This technique has been used before [11, 19]. We use $\mathcal{P}$ to denote the (possibly extended) input PLC.

Incidence. Two elements in $\mathcal{P}$ are incident if one is in the boundary of the other.

Adjacent elements. We call two elements of $\mathcal{P}$ *adjacent* if either they are incident or they are nonincident but their closure intersect. For example, two segments sharing an endpoint are adjacent, and two facets sharing a segment are adjacent. As another example, if a segment does not lie on the boundary of a facet but they share a vertex, then they are adjacent. A vertex of $\mathcal{P}$ is not adjacent to any other element that is not incident to it.

Local feature size. The local feature size for $\mathcal{P}$ is a function $f : \mathbb{R}^3 \to \mathbb{R}$, where $f(x)$ is the radius of the smallest ball centered at x intersecting two nonadjacent elements of $\mathcal{P}$.

Weighted Delaunay triangulation. We use $\widehat{x}$ to denote a *weighted point* at x with weight X^2. The weighted point $\widehat{x}$ can be interpreted as a sphere centered at x with radius X. Notice that any point can be thought of as a weighted point with weight zero. The *weighted distance* $\pi(\widehat{x}, \widehat{y})$ between two weighted points $\widehat{x}$ and $\widehat{y}$ is given by

$$\pi(\widehat{x}, \widehat{y}) = \|x - y\|^2 - X^2 - Y^2.$$

If $\pi(\widehat{x}, \widehat{y}) = 0$, then for any point $z \in \widehat{x} \cap \widehat{y}$, $\|x-z\|^2 + \|y-z\|^2 = X^2 + Y^2 = \|x-y\|^2$. That is, $\angle xzy = \pi/2$. Thus we say that $\widehat{x}$ and $\widehat{y}$ are *orthogonal.* If $\pi(\widehat{x}, \widehat{y})$ is greater (resp., smaller) than 0, then for any point $z \in \widehat{x} \cap \widehat{y}$, $\angle xzy > \pi/2$ (resp., $\angle xzy < \pi/2$), and we say that $\widehat{x}$ and $\widehat{y}$ are *further than (resp., closer to) orthogonal* from each other. The *bisector plane* of $\widehat{x}$ and $\widehat{y}$ is the locus of points at equal weighted distances from $\widehat{x}$ and $\widehat{y}$.

Let τ be a simplex of dimension one or more in $\mathbb{R}^3$; i.e., τ is an edge, a triangle, or a tetrahedron. The *smallest orthosphere* of τ is the smallest sphere, say $\widehat{x}$, so that $\widehat{x}$ is orthogonal to each weighted vertex of τ. The smallest orthosphere is the counterpart of the *smallest circumspheres* for simplices with unweighted vertices; see Figure 2.1. Notice that for a tetrahedron there is only a single sphere orthogonal to all of its four weighted vertices, which is its smallest orthosphere. The center and radius of the smallest orthosphere of any simplex are called its *orthocenter* and *orthoradius*, respectively.

For a weighted point set $\mathcal{V}$, a tetrahedron spanning four points of $\mathcal{V}$ is *weighted Delaunay* if its orthosphere is further than orthogonal away from any other weighted point in $\mathcal{V}$. A *weighted Delaunay triangulation* of $\mathcal{V}$ is the collection of all weighted Delaunay tetrahedra along with their triangles, edges, and vertices.

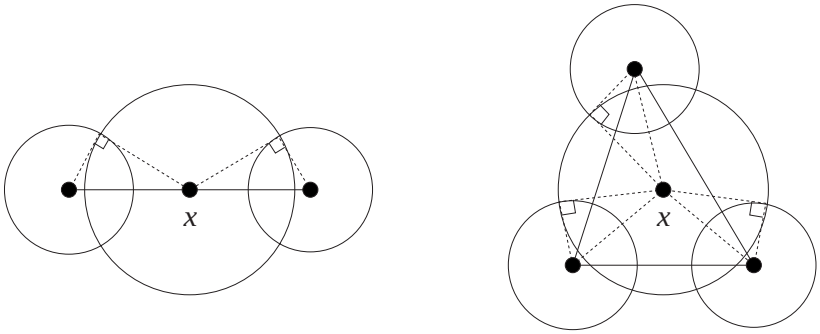


FIG. 2.1. *Smallest orthospheres of an edge and a triangle with orthocenter x.*

3. Weighted Delaunay refinement.

3.1. Overview. The Delaunay refinement algorithm as originally proposed by Ruppert [19] and later extended to three dimensions by Shewchuk [20] iteratively inserts circumcenters of tetrahedra that have radius-edge ratio above a threshold. Whenever a circumcenter x lies so close to an element F in the input PLC that some of its subsets cannot appear in the current Delaunay triangulation, x is rejected and F is subdivided instead. It can be proved that a new vertex x is inserted at least $c \cdot f(x)$ distance away from all other vertices and input elements for some constant $c > 0$. This lower bound on distances guarantees the termination because only a finite number of points can be accommodated in a bounded domain with a lower bound on the interpoint distances.

Delaunay refinement achieves bounded radius-edge ratio but fails to remove slivers. This motivated the sliver exudation method of Cheng et al. [7], which eliminates slivers from a Delaunay mesh that already have bounded radius-edge ratio. The key algorithmic tool in sliver exudation is the assignment of weights to unweighted vertices. The weight assignment can be viewed as *pumping* the unweighted vertex to grow it to a sphere (weighted vertex). When an unweighted vertex v is pumped, there is a restriction on the weight to be assigned to v. The weight must be selected from the interval $[0, \omega^2 N(v)^2]$, where $N(v)$ is the Euclidean distance to its nearest vertex and $\omega \in (0, 1/2)$ is a constant. It is shown that there exists a weight in the mentioned interval for each vertex v so that if v is assigned that weight, all slivers incident to v are removed from the weighted Delaunay triangulation of the vertex set (see [7]). This is stated precisely in the following sliver theorem.

THEOREM 3.1 (sliver theorem; see [7]). *Given a periodic point set $\mathcal{V}$ and a Delaunay triangulation of $\mathcal{V}$ with radius-edge ratio $\leq \rho$, there exist $\rho_0 > 0$, $\sigma_0 > 0$, and a weight assignment in $[0, \omega^2 N(v)^2]$ for each vertex v in $\mathcal{V}$ such that $\rho(\tau) \leq \rho_0$ and $\sigma(\tau) > \sigma_0$ for each tetrahedron τ in the weighted Delaunay triangulation of $\mathcal{V}$.*

The required weights can be assigned in a deterministic manner for periodic point sets as defined by Cheng et al. [7]. Periodic point sets are infinite points sets without boundaries. So Theorem 3.1 does not give an algorithm for meshing bounded domains. For a bounded domain, the weight assignment may challenge the boundary. We solve this problem by combining the Delaunay refinement with pumping while redefining the encroachment with respect to weighted points.

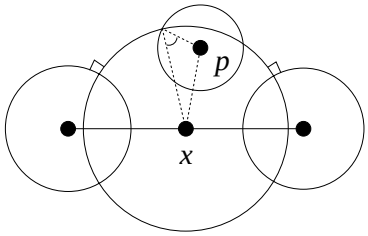


FIG. 3.1. *A subsegment is encroached by $\widehat{p}$. Note that the angle shown between $\widehat{p}$ and the smallest orthosphere of the subsegment is less than $\pi/2$. The subsegment is split by its orthocenter x.*

Our algorithm refines a Delaunay triangulation until it determines that it is safe to pump vertices to remove slivers. In the refinement process it attempts to insert circumcenters of skinny tetrahedra. However, these centers may come close to or challenge some boundary elements, which are then subdivided. The subdivision process may trigger further subdivision with new vertices. This refinement process is exactly the same as that of Ruppert [19] and Shewchuck [20]. We introduce another stimulus for refinement in preparation for pumping the vertices to remove slivers in a final stage. If a vertex v has a sliver incident to it, we check whether the vertex with maximum allowed weight challenges any boundary element. If so, a refinement process is triggered. We will see that even if v does not challenge any boundary element, $\widehat{v}$, the weighted vertex, may. At these stages, we maintain only the unweighted Delaunay triangulation of the current vertex set. At the end of the refinement process, when no further boundary element is challenged by weighted or unweighted vertices, we safely pump the vertices to eliminate slivers.

3.2. Subsegments and subfacets. Our algorithm maintains a set of vertices $\mathcal{V}$ which consists of the input vertices initially, and it grows as we refine boundary elements and insert circumcenters of skinny tetrahedra.

The vertices in $\mathcal{V}$ on a segment of $\mathcal{P}$ subdivide it into *subsegments*. Let ab be a subsegment. A point p *encroaches* upon ab if p lies inside the smallest circumsphere of ab. Suppose that there is no encroached subsegment. Consider the vertices in $\mathcal{V}$ on a facet of $\mathcal{P}$ in isolation. The two-dimensional Delaunay triangulation of these vertices conforms to the boundary of the facet. Each triangle on the facet is called a *subfacet*. A point p *encroaches* upon a subfacet abc if p lies inside the smallest circumsphere of abc.

Eventually, we will assign weights to vertices in $\mathcal{V}$. This will require us to deal with the weighted versions of subsegments, subfacets, and encroachment. The *weighted-subsegments* are exactly the same as subsegments but possibly with weighted endpoints. A weighted point $\widehat{p}$ encroaches upon a weighted-subsegment ab if $\widehat{p}$ is closer than orthogonal to the smallest orthosphere of ab. See Figure 3.1 for an illustration. Suppose that there is no encroached weighted-subsegment. Consider the weighted vertices on a facet of $\mathcal{P}$ in isolation. The two-dimensional weighted Delaunay triangulation of these vertices conforms to the boundary of the facet. Each triangle on the facet is called a *weighted-subfacet*. A weighted point $\widehat{p}$ encroaches upon a weighted-subfacet abc if $\widehat{p}$ is closer than orthogonal to the smallest orthosphere of abc.

We will prove several properties for the weighted Delaunay triangulation (Lemmas 3.2–3.6) later. As the weighted case is more general than the unweighted case, these results are applicable for the unweighted Delaunay triangulation as well as when only some of the vertices are weighted.

3.3. Weight assignment. For a vertex u, there are at most two assigned weights, one to check whether $\widehat{u}$ encroaches upon a boundary element, and possibly another if u participates in an actual pumping in the final stage. Let $\omega_0 < 1$ be a constant chosen in advance. The value of ω_0 will be determined in section 5. We use the weight $\omega_0^2 f(u)^2$ for an encroachment check and the interval $[0, \omega_0^2 f(u)^2]$ to assign weights during pumping. It is important that there be enough space around u when it is pumped. Recall that the vertex $\widehat{u}$ with weight U^2 is equivalent to a sphere centered at u with radius U. The assigned weights also impose a requirement that no other vertex be allowed within an Euclidean distance of $2\omega_0 f(u)$ from u. Consequently, after pumping, $\widehat{u}$ can reach up to half of the Euclidean distance to its nearest neighbor. Thus no two weighted vertices intersect after pumping. We provide an exact statement of this property below.

Vertex gap property. For each vertex u in $\mathcal{V}$, the weight of u used for encroachment checking or pumping is at most $\omega_0^2 f(u)^2$, and the Euclidean nearest neighbor distance of u in $\mathcal{V}$ is at least $2\omega_0 f(u)$.

Note that we need to maintain the vertex gap property throughout the algorithm as new vertices are inserted (added to $\mathcal{V}$). As the weights assigned to the vertices are not large compared with interpoint distances, the resulting weighted Delaunay triangulation satisfies many properties of the unweighted one, and many results of the Delaunay refinement carry over to the weighted Delaunay refinement (Lemmas 3.2–3.6). We will prove the vertex gap property in section 5.

3.4. Locations of centers. Whenever a subsegment is encroached, we split it by inserting its midpoint. Whenever a subfacet is encroached, we split it by inserting its circumcenter. This requires the circumcenter to lie on the facet of $\mathcal{P}$ containing that subfacet. Whenever there is a skinny tetrahedron, we insert its circumcenter. This requires the circumcenter to lie inside the input domain to prevent perpetual growth of the mesh. Although at this point we need these results for Delaunay triangulations, we prove them for weighted Delaunay triangulations that we will need in section 7 for pumping.

LEMMA 3.2. *Suppose that the vertex gap property holds. If there is no encroached weighted-subsegment or weighted-subfacet, no weighted vertex $\widehat{p}$ intersects a segment or a facet that does not contain p.*

Proof. Assume to the contrary that the lemma does not hold. First of all, $\widehat{p}$ cannot enclose any vertex other than p because of the vertex gap property. Let F be a segment intersected by $\widehat{p}$ if there is any. Otherwise, let F be a facet intersected by $\widehat{p}$. Let q be the projection of p on F. Observe that any point on F lies inside the smallest orthosphere of some weighted-subsegment or weighted-subfacet, or inside some weighted vertex on F (viewed as a sphere). Suppose that q lies inside the smallest orthosphere $\widehat{x}$ of a weighted-subsegment or weighted-subfacet τ. We have $\|p - x\|^2 = \|q - x\|^2 + \|p - q\|^2 < X^2 + P^2$ as $\|q - x\| < X$ and $\|p - q\| < P$. This implies that $\widehat{p}$ encroaches upon τ, a contradiction. Suppose that q lies inside a weighted vertex $\widehat{v}$. Similarly, we get $\|p - v\|^2 = \|q - v\|^2 + \|p - q\|^2 < V^2 + P^2$. This implies that $\widehat{p}$ and $\widehat{v}$ intersect, which contradicts the vertex gap property. ∎

LEMMA 3.3. *Suppose that the vertex gap property holds.*

(i) *A weighted-subsegment contains its orthocenter.*
(ii) *If no weighted-subsegment is encroached, a facet contains the orthocenter of any weighted-subfacet on it.*
(iii) *If no weighted-subsegment or weighted-subfacet is encroached, the input domain contains the orthocenter of any weighted Delaunay tetrahedron inside the input domain.*

Proof. Because of the vertex gap property, (i) is obvious. We present a proof that works for both (ii) and (iii). Let Ω be a facet or the input domain. Let τ be a weighted-subfacet on Ω or a weighted Delaunay tetrahedron inside Ω correspondingly. Let $\widehat{s}$ be the smallest orthosphere of τ. Assume to the contrary that s lies outside Ω.

Let p be a vertex of τ such that ps crosses $\partial\Omega$. By Lemma 3.2, $\widehat{p}$ does not cross $\partial\Omega$, so $\widehat{s}$ must cross $\partial\Omega$ in order to be orthogonal to $\widehat{p}$. Let E be the element in $\partial\Omega$ closest to s. As $\widehat{s}$ is empty of vertices, E is either a segment or facet. Let L_E be the affine hull of E. Let $\widehat{y}$ be the diametral/equatorial sphere of $\widehat{s} \cap L_E$. We make four observations (see Figure 3.2).

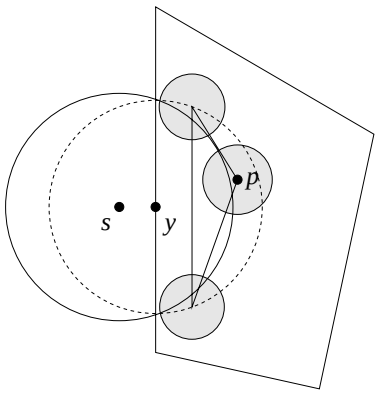


FIG. 3.2. *Illustration for Lemma* 3.3 *when* Ω *is a facet. The shaded disks are the vertices of the weighted-subfacet whose orthocenter is* s*. The solid circle is* $\widehat{s}$*, and the dashed one is* $\widehat{y}$*.*

First, y lies in the interior of E.

Second, the bisector plane of $\widehat{s}$ and $\widehat{y}$ contains E. Note that p and s lie on opposite sides of this bisector plane. So $\pi(\widehat{p}, \widehat{y}) < \pi(\widehat{p}, \widehat{s}) = 0$.

Third, for any vertex v on E, we have $\pi(\widehat{v}, \widehat{y}) \geq 0$ because $\pi(\widehat{v}, \widehat{s}) = \pi(\widehat{v}, \widehat{y})$ and $\widehat{s}$ is not closer than orthogonal to $\widehat{v}$.

Fourth, we claim that the projection q of p onto L_E lies in the interior of E. Assume to the contrary that q does not lie in the interior of E. By the first observation, y lies in the interior of E. So qy intersects an endpoint of E if L_E is a line or qy intersects a weighted-subsegment in ∂E if L_E is a plane. Let γ denote the endpoint/weighted-subsegment that qy intersects. Let $\widehat{x}$ denote $\widehat{\gamma}$ if γ is a vertex, or the smallest orthosphere of γ if γ is a weighted-subsegment. Let H be the bisector plane of $\widehat{x}$ and $\widehat{y}$. Consider $\pi(\widehat{v}, \widehat{x})$ for each vertex v of γ. If γ is a vertex, then $\pi(\widehat{v}, \widehat{x}) = \pi(\widehat{x}, \widehat{x}) = -2X^2 \leq 0$; otherwise, γ is a weighted-subsegment and $\pi(\widehat{v}, \widehat{x}) = 0$. On the other hand, by the third observation, $\pi(\widehat{v}, \widehat{y}) \geq 0$ for each vertex v of γ. We conclude that for each vertex v of γ, $\pi(v, \widehat{y}) \geq \pi(v, \widehat{x})$. Thus γ lies in the halfspace H^+ bounded by H, where $\pi(a, \widehat{x}) \leq \pi(a, \widehat{y})$ for each point $a \in H^+$. The ray emitting from x through y must shoot outside H^+ because a point sufficiently far in this direction is closer to $\widehat{y}$ than $\widehat{x}$. Coupling this with the fact that qy intersects γ, we conclude that $q \in H^+$. Note that pq is parallel to H. Thus $p \in H^+$, which implies that $\pi(\widehat{p}, \widehat{x}) \leq \pi(\widehat{p}, \widehat{y})$. By the second observation, $\pi(\widehat{p}, \widehat{y}) < 0$ and so $\pi(\widehat{p}, \widehat{x}) < 0$. If γ is a vertex, this contradicts the vertex gap property; otherwise, it contradicts the assumption that no weighted-subsegment is encroached. This proves the claim.

Let τ be the weighted-subsegment or weighted-subfacet in the interior of E that contains q. Let $\widehat{z}$ be the smallest orthosphere of τ. Let H_1 be the bisector plane of $\widehat{y}$

and $\widehat{z}$. By the third observation, for each vertex v of τ, $\pi(\widehat{v},\widehat{y}) \geq 0 = \pi(\widehat{v},\widehat{z})$. Thus τ lies inside the halfspace H_1^+ bounded by H_1, where $\pi(a,\widehat{z}) \leq \pi(a,\widehat{y})$ for each point $a \in H_1^+$. As $q \in \tau$ and pq is parallel to H_1, we have $p \in H^+$. Thus $\pi(\widehat{p},\widehat{z}) \leq \pi(\widehat{p},\widehat{y})$, which is negative by the second observation. However, $\widehat{p}$ encroaches upon τ then, a contradiction. □

3.5. Adjacent elements and encroachment. A key property in ordinary Delaunay refinement (without weight assignment) is that vertices on one element of $\mathcal{P}$ cannot encroach upon subsegments and subfacets on an adjacent element of $\mathcal{P}$, provided that no input angle is less than $\pi/2$. This property is needed for the algorithm to terminate. We show that this property also holds for the weighted case.

LEMMA 3.4. *If the vertex gap property holds, then for any weighted-subsegment ab on an edge e of $\mathcal{P}$, ab cannot be encroached upon by any vertex that lies on an edge adjacent to e or a facet adjacent but nonincident to e.*

Proof. Let $\widehat{x}$ be the smallest orthosphere of ab. By Lemma 3.3(i), x lies on ab. Let E be an edge adjacent to e or a facet adjacent but nonincident to e. Let u be a vertex on E. Let v be the common vertex of E and e. By the vertex gap property, $\widehat{u}$ does not contain v. Clearly, $\widehat{x}$ does not contain v. $\pi(\widehat{x},\widehat{u}) = \|u-x\|^2 - X^2 - U^2 > \|u-x\|^2 - \|v-x\|^2 - \|u-v\|^2$. Because all input angles are at least $\pi/2$, $\|u-x\|^2 \geq \|v-x\|^2 + \|u-v\|^2$. Thus $\pi(\widehat{x},\widehat{u}) > 0$ and $\widehat{u}$ does not encroach upon ab. □

LEMMA 3.5. *Let abc be a weighted-subfacet on a facet F of $\mathcal{P}$. If there is no encroached weighted-subsegment, then abc cannot be encroached by any vertex that lies on a facet adjacent to F or an edge adjacent but non-incident to F.*

Proof. Let H be the plane containing F. Let T denote the two-dimensional weighted Delaunay triangulation of the vertices on F. Let V_T denote the subdivision of H induced by T. Let Σ be the set of the smallest orthospheres of the triangles in T, the smallest orthospheres of weighted-subsegments in ∂F, and a sphere centered at infinity in H with infinite radius. By duality, V_T is the intersection of H and the weighted Voronoi diagram of Σ. Let u be a vertex that lies on a facet adjacent to F or an edge adjacent but nonincident to F. Because no input angle is less than $\pi/2$, the orthogonal projection of u onto H falls outside F or on ∂F. Consider any weighted-subfacet abc on F. The directed segment from the projection of u to a intersects a sequence of cells of V_T, including some weighted-subsegment vw in ∂F. This yields a corresponding sequence of spheres in Σ owning the cells intersected. The weighted distances from $\widehat{u}$ to the spheres in the sequence increase along the sequence. It follows that if $\widehat{u}$ encroaches upon abc, $\widehat{u}$ also encroaches upon vw, a contradiction. □

3.6. Projection. Instead of finding any encroached subfacet, we will focus on one that contains the projection of its encroaching vertex. Such a result has been proved by Shewchuk [20] for encroachments by unweighted points in the unweighted Delaunay triangulations. We will need a weighted version of this result.

LEMMA 3.6. *If no weighted-subsegment is encroached and $\widehat{p}$ encroaches upon some weighted-subfacet on a facet F, then there exists a weighted-subfacet h on F which is encroached upon by $\widehat{p}$, and h contains the orthogonal projection of p on F.*

Proof. Let H be the plane containing F. Let T denote the two-dimensional weighted Delaunay triangulation of the vertices on F. Let V_T denote the subdivision of H induced by T. Let Σ be the set of the smallest orthospheres of the triangles of T, the smallest orthospheres of the subsegments in ∂F, and a sphere centered at infinity in H with infinite radius. V_T is the intersection of H and the weighted Voronoi diagram of Σ. Let t_1 be a weighted-subfacet of F that is encroached upon by $\widehat{p}$. Suppose that p projects to a cell t_2 in V_T. When we walk along the directed segment

from the projection of p (inside t_2) to an interior point of t_1, we encounter a sequence of cells in V_T. This yields a corresponding sequence of spheres in Σ owning the cells encountered. The weighted distances from $\widehat{p}$ to the spheres in this sequence increase along the sequence. If t_2 is outside F, then the walk will exit a triangle t outside F and enter a triangle t' inside F at some point; i.e., $t \cap t'$ is a weighted-subsegment in ∂F. Because $\widehat{p}$ encroaches upon t_1, we have $\pi(\widehat{p}, \widehat{x}) \leq \pi(\widehat{p}, \widehat{y}) < 0$, where $\widehat{x}$ is the smallest orthosphere of $t \cap t'$ and $\widehat{y}$ is the smallest orthosphere of t_1. It follows that $\widehat{p}$ encroaches upon $t \cap t'$, which is a contradiction. □

3.7. Algorithm. The input to our algorithm QualMesh is a PLC. The PLC bounds a convex domain and may contain vertices, segments, and facets within the domain. We also assume that no input angle is less than $\pi/2$. This includes all angles between two segments sharing a vertex, a segment and a facet sharing one vertex, or two facets sharing a vertex or a segment. The assumption of a convex bounded domain is not a serious restriction because any PLC can be enclosed within a large enough box whose elements are included in the extended PLC. After the meshing is finished, one can choose to retain the desired tetrahedra. This standard technique has been used before [11, 19].

In the algorithm below we have a refinement step which is done in the unweighted Delaunay triangulation though some of the refinements may be triggered by a weighted point. Subsequent to this refinement, the vertices are pumped to eliminate slivers. Obviously, this is carried out in the weighted Delaunay triangulation. The results proved so far for the weighted Delaunay triangulation also remain valid for the unweighted case (where all weights are assumed to be zero) as well as when only some vertices are weighted.

QualMesh($\mathcal{P}$).

1. Compute the Delaunay triangulation of the input vertices of $\mathcal{P}$.
2. Repeatedly apply a rule from the following list until no rule is applicable. Rule i is applied only if it is applicable and no Rule j with $j < i$ is applicable. The parameters ρ_0, σ_0, and ω_0 will be determined later.

   Rule 1 (subsegment refinement). If there is an encroached subsegment, insert its midpoint.

   Rule 2 (subfacet refinement). If there is an encroached subfacet, there exists an encroached subfacet h that contains the projection of its encroaching vertex on the facet containing h. Insert the circumcenter of h, provided that it does not encroach upon any subsegment. Otherwise, reject the circumcenter and apply Rule 1 to split an encroached subsegment.

   Rule 3 (tetrahedron refinement). Assume that there is a tetrahedron with radius-edge ratio exceeding ρ_0 and circumcenter z. If z does not encroach upon any subsegment or subfacet, insert z. Otherwise, reject z and perform one of the following actions:
   - If z encroaches upon some subsegment(s), use Rule 1 to split one such subsegment.
   - Otherwise, z encroaches upon some subfacet(s) and use Rule 2 to split one such subfacet that contains the projection of z.

   Rule 4 (weighted encroachment). Let $\operatorname{Del}\mathcal{V}$ be the Delaunay triangulation of the current vertex set $\mathcal{V}$. Take a vertex v that is incident on a sliver τ (i.e., $\sigma(\tau) \leq \sigma_0$). Let $\widehat{v}$ be the weighted vertex v with weight $\omega_0^2 f(v)^2$.
   - If $\widehat{v}$ encroaches upon some subsegment in $\operatorname{Del}\mathcal{V}$ that does not lie on

the same segment as v, use Rule 1 to split one such subsegment.
- If $\widehat{v}$ encroaches upon some subfacet in $\mathrm{Del}\,\mathcal{V}$ that does not lie on the same facet as v, use Rule 2 to split one such subfacet that contains the projection of v.

Notice that we always maintain an unweighted Delaunay triangulation in step 2, and $\widehat{v}$ is used only for checking encroachments.

3. For each vertex v incident on a sliver τ (i.e., $\sigma(\tau) \leq \sigma_0$), pump v with weight in $[0, \omega_0^2 f(v)^2]$ until no sliver is incident to v. Maintain the weighted Delaunay triangulation during the pumping. We claim that no pumped vertex encroaches upon any weighted-subsegment and weighted-subfacet.

3.8. Time analysis. We analyze the time complexity of the algorithm in terms of n, the number of input vertices, and N, the number of output vertices. We will prove in section 7.4 that N is no more than a constant factor of the minimum number of vertices possible.

Consider the lifting map $\mu : \mathbb{R}^3 \to \mathbb{R}^4$, which maps a point $x = (x_1, x_2, x_3) \in \mathbb{R}^3$ to a point $\mu(x) = (x_1, x_2, x_3, x_1^2 + x_2^2 + x_3^2) \in \mathbb{R}^4$. For a point set $\mathcal{V}$ in three dimensions, let $\mu(\mathcal{V}) = \{\mu(v),\ v \in \mathcal{V}\}$. The Delaunay triangulation of $\mathcal{V}$ is the projection of the convex hull of $\mu(\mathcal{V})$ (see [10]). The first step can be done in $O(n^2)$ time using Chazelle's convex hull algorithm [4]. The second step is a loop, and a new vertex is added in each iteration. Thus there are fewer than N iterations. After each vertex insertion in Rule 1, 2, 3, or 4, we have to update the Delaunay triangulation. An inserted point p is the center of a circumsphere, circumcircle, or a segment. In each case we get a tetrahedron that is destroyed after inserting p. We explore in the Delaunay data structure in a depth-first manner to collect all tetrahedra that are destroyed with the insertion of p. Once these tetrahedra are identified, p is connected to the boundary of the union of them to update the Delaunay triangulation. If D_p is the number of deleted tetrahedra, the complexity of this update is $O(D_p)$. Thus the total time of all updates over the entire algorithm is upper bounded by the number of deleted tetrahedra. We argue that this number is $O(N^2)$.

In the lifted diagram in four dimensions, the insertion of p can be viewed as follows. The point $\mu(p)$ is below the convex hull of $\mu(\mathcal{V})$, and let T be the set of tetrahedra on this convex hull visible to $\mu(p)$. Insertion of $\mu(p)$ destroys all tetrahedra in T and creates new tetrahedra on the updated convex hull by connecting $\mu(p)$ to the boundary of the union of tetrahedra in T. Let us call the union of new tetrahedra incident to $\mu(p)$ its *cap*. The space between the cap of $\mu(p)$ and T can be triangulated by connecting $\mu(p)$ to each tetrahedron in T. Thus, assuming that the convex hull of the initial point set is triangulated, one can maintain a triangulation in the lifted diagram after each insertion, which contains the lifted deleted tetrahedra. Therefore, all tetrahedra deleted by QualMesh can be mapped to tetrahedra in the triangulation of N points in four dimensions. Since the size of any triangulation of N points in four dimensions is only $O(N^2)$ (see Theorem 1.2 of [9]), the same bound applies to the number of deleted tetrahedra.

Each insertion is preceded by a search of an encroached subsegment, subfacet, or skinny tetrahedron. We argue that this search can also be done in $O(N^2)$ total time. We maintain a stack of all skinny tetrahedra. Whenever an update is performed in the triangulation, we update the stack and mark any tetrahedron deleted through pointers. Thus, a skinny tetrahedron can be obtained by popping the stack until the popped tetrahedron is not marked. The time to create the initial stack is $O(n^2)$, the complexity of the initial Delaunay triangulation. The time to update the stack can be

absorbed in the triangulation update time, which is $O(N^2)$ in total. Next, we need to account for searching the encroached subsegments and subfacets. This encroachment may occur by an inserted or rejected point. Since each rejected point leads to an insertion, the total number of inserted and rejected points is $O(N)$. For each such point we can scan all subfacets and subsegments to determine the encroachments. At any time of the algorithm, the total number of subsegments and subfacets is $O(N)$. This is because the subfacets and subsegments on a planar facet create a plane graph whose complexity is linear in terms of the number of vertices, and each input edge can be incident to only a constant number of facets due to the input angle constraint. Therefore, counting over all points, all encroachments can be determined in $O(N^2)$ time.

Other than vertex insertion, we also need to compute $f(v)$ for some vertex v in each application of Rule 4. This can be done in $O(n)$ time by checking all the input elements. Thus the total time spent in computing local feature sizes is $O(Nn)$. Hence, the total time taken by the second step is $O(N^2 + Nn + n^2) = O(N^2)$. The third step pumps at most N vertices. Each pumping requires updating the mesh connectivity. The complexity of these updates again can be counted through the lifted diagram to be $O(N^2)$. Thus, the third step takes $O(N^2)$ time. In all, we have the following theorem.

THEOREM 3.7. *The time complexity of* QUALMESH *is* $O(N^2)$*, where* N *is the minimum number of points required to mesh the input domain with tetrahedra that have bounded aspect ratio.*

4. Insertion radii. We define a notion of *insertion radius* for each vertex inserted or rejected by QUALMESH. The main result in this section is a relation among insertion radii and local feature sizes which allows us to prove a lower bound on intervertex distances in section 5. For any vertex x in the input PLC, the insertion radius r_x is the Euclidean distance from the nearest input vertex. For any vertex x that QUALMESH inserts or rejects, the insertion radius r_x is the distance of x from the nearest vertex in the current $\mathcal{V}$. In section 5, we will see that a lower bound on the insertion radii of vertices implies a lower bound on the intervertex distances. Thus a packing argument shows that QUALMESH must terminate as the domain has bounded volume.

The analysis uses a parent-child relation among all vertices that are input vertices or inserted/rejected by QUALMESH. This is similar to the parent-child relation defined by Shewchuk [20] for the three-dimensional Delaunay refinement, but we need some modifications because of refinement triggered by weighted points. Parents of input vertices are undefined. If QUALMESH inserts or rejects a vertex x using Rule i, $1 \le i \le 3$, then x has type i. The parent of x is defined as follows.

1. x has type 3. Among the two endpoints of the shortest edge of the tetrahedron split by x, the vertex p that appeared in the latest $\mathcal{V}$ is the parent of x.
2. x has type 1 or 2. Then x is the circumcenter of a subsegment or subfacet τ. There are two cases.
   (a) If τ is encroached by a weighted vertex $\widehat{p}$ in Rule 4, the parent of x is p.
   (b) If τ is encroached by an unweighted vertex, we choose the parent of x to be the encroaching vertex p nearest to x. If p was not rejected, then $r_x = \|p - x\|$; otherwise, $r_x = X \ge \|p - x\|$.

Our goal is to lower bound r_x in terms of $f(x)$ and r_p, where p is the parent of x. (This recurrence will be useful in an inductive proof to lower bound r_x in terms of $f(x)$ only.) To this end, we prove two technical lemmas. Lemma 4.1 lower bounds

$\|p-x\|$ in terms of $f(x)$, $f(p)$, and r_p. Lemma 4.2 lower bounds r_x in terms of $\|p-x\|$. Both apply under some special conditions. Then we employ these two lemmas and analyze the remaining cases to obtain the lower bound on r_x in terms of $f(x)$ and r_p.

LEMMA 4.1. *Suppose that the vertex gap property holds. Let x be a vertex of type 1 or 2. Let p be the parent of x. Let $\widehat{x}$ be the smallest circumsphere of the subsegment or subfacet centered at x.*

(i) *If p is an input vertex, p has type 1, or p has type 2 and x has type 2, then $\|p-x\| \geq \max\{f(x), f(p)\}$.*

(ii) *Suppose that p has type 2 and x has type 1, or that p has type 3. If p does not lie inside $\widehat{x}$, then $\|p-x\| \geq r_p/\sqrt{2}$.*

Proof. Let F be the segment containing x if x has type 1; otherwise let F be the facet containing x. We first claim that p does not lie on F. If the insertion/rejection of x is induced in Rule 4, the claim is enforced by the algorithm. Otherwise, because the unweighted Delaunay triangulation of vertices on a segment or facet gives the subsegments and subfacets, an unweighted vertex on a segment (facet) cannot encroach upon subsegments (subfacets) on the same segment (facet).

Case 1: p is an input vertex. The two balls centered at p and x with radius $\|p-x\|$ intersect two nonadjacent input elements (F and p). Thus $\|p-x\| \geq f(p)$ and $\|p-x\| \geq f(x)$.

Case 2: p has type 1, or p has type 2 and x has type 2. Let F' be the segment/facet containing p. We already know that $F' \neq F$. We argue that F and F' are nonadjacent. If x has type 1 and p has type 1, then Lemma 3.4 implies that F' is nonadjacent to F. Suppose that x has type 2. The insertion/rejection of x could be induced in an application of Rule 4, or it could be induced in a direct application of Rule 2. In any case, there is no encroached subsegment; otherwise Rule 1 would have been invoked instead. Thus Lemma 3.5 implies that F' and F are nonadjacent. As F and F' are nonadjacent, $\|p-x\| \geq f(x)$ and $\|p-x\| \geq f(p)$.

Case 3: p has type 2 and x has type 1, or p has type 3. (When p has type 2 and x has type 1, p and x may lie on the same facet.) By assumption, p does not lie inside $\widehat{x}$. This implies that the insertion/rejection of x is induced in Rule 4. Also, QUALMESH enforces two conditions. First, p was inserted by QUALMESH. Second, if x has type 1 (resp., type 2) and lies on a segment (resp., facet), p does not lie on the same segment (resp., facet). It suffices to lower bound the distance from p to F. Go back to the time when p was inserted by QUALMESH.

Case 3.1: x has type 1. Then F is a segment. Let ab be the subsegment on F that contains x at that time. Observe that p lies outside the smallest circumsphere of ab; otherwise, p would have been rejected for encroaching upon a subsegment because p has type 2 or 3. If the nearest point of ab to p is a or b, then $\|p-x\| \geq \min\{\|p-a\|, \|p-b\|\} \geq r_p$ because a and b exist when p is inserted. Otherwise, the nearest point lies in the interior of ab. Assume that the orthogonal projection of p onto ab lies on ay, where y is the midpoint of ab. Then $\sin\angle pay \geq 1/\sqrt{2}$. Thus $\|p-x\| \geq \|a-p\| \cdot \sin\angle pay \geq r_p/\sqrt{2}$.

Case 3.2: x has type 2. Then p has type 3 and F is a facet. Observe that p lies outside the smallest circumsphere of any subsegment in ∂F or any subfacet on F. Otherwise, p would have been rejected for encroaching upon a subsegment or subfacet because p has type 3.

If the nearest point of F to p lies on a segment F' in ∂F, by apply-

ing the analysis in Case 3.1 to F', we conclude that $\|p-x\| \geq r_p/\sqrt{2}$. Otherwise, the nearest point q on F to p lies inside some subfacet abc on F. Note that q is the orthogonal projection of p onto abc. The Voronoi diagram of a, b, and c divides abc into three regions, each containing a vertex of abc. Assume that q lies inside the region containing a. Let $\widehat{z}$ be the smallest circumsphere of abc. Let H be the plane that is perpendicular to F and passes through a and p. $H \cap \widehat{z}$ is a circle and p lies outside it. Let y be the center of $H \cap \widehat{z}$. Observe that q lies on ay and so $\sin \angle pay \geq 1/\sqrt{2}$. The Euclidean distance from p to F is $\|a-p\| \cdot \sin \angle pay \geq r_p/\sqrt{2}$. □

LEMMA 4.2. *Suppose that the vertex gap property holds. Let x be a vertex of type 1 or 2. Let p be the parent of x. Then $r_x \geq \|p-x\|/\sqrt{2}$.*

Proof. Let $\widehat{x}$ be the circumsphere of which x is the center. If p lies inside $\widehat{x}$, p is unweighted, and it follows from the definition of parent that $r_x \geq \|p-x\|$. Assume that p does not lie inside $\widehat{x}$. For p to be the parent of x, the insertion/rejection of x must be induced in Rule 4 by the weighted vertex $\widehat{p}$ with weight $\omega_0^2 f(p)^2$. This also implies that $p \in \mathcal{V}$ at that time. By Lemma 4.1, $\|p-x\| \geq f(p)$ or $\|p-x\| \geq r_p/\sqrt{2}$. If $\|p-x\| \geq f(p)$, then $\|p-x\|/\sqrt{2} \geq \omega_0 f(p)$ as $\omega_0 \leq 1/2$. If $\|p-x\| \geq r_p/\sqrt{2}$, as $r_p \geq 2\omega_0 f(p)$ by the vertex gap property, we also get $\|p-x\|/\sqrt{2} \geq \omega_0 f(p)$. Because $\widehat{p}$ was closer than orthogonal to $\widehat{x}$, we have $X^2 + \omega_0^2 f(p)^2 > \|p-x\|^2$. Substituting $\omega_0^2 f(p)^2$ by $\|p-x\|^2/2$ and rearranging terms, we get $r_x = X > \|p-x\|/\sqrt{2}$. □

We are ready to lower bound r_x in terms of $f(x)$ and the insertion radius of the parent of x. This is the main result of this subsection.

LEMMA 4.3. *Suppose that the vertex gap property holds. Let x be an input vertex or a vertex inserted or rejected. Let p be the parent of x, if it exists.*

(i) *If x is an input vertex or p is an input vertex, then $r_x \geq f(x)/\sqrt{2}$.*
(ii) *Otherwise, $r_x \geq f(x)/\sqrt{2}$ or $r_x \geq c \cdot r_p$, where $c = 1/2$ if x has type 1 or 2 and $c = \rho_0$ if x has type 3.*

Proof. If x is an input vertex, then $r_x \geq f(x)$ by the definition of local feature size. Suppose that QUALMESH inserts x or rejects x by Rule 1, 2, or 3. Recall that x is the center of the smallest circumsphere of a subsegment, subfacet, or tetrahedron. Let $\widehat{x}$ denote this circumsphere.

Consider the case where p is an input vertex. If x has type 1 or 2, then $\|p-x\| \geq f(x)$ by Lemma 4.1. Note that $r_x \geq \|p-x\|/\sqrt{2}$ by Lemma 4.2. Thus we get $r_x \geq f(x)/\sqrt{2}$. Suppose that x has type 3. Let τ be the skinny tetrahedron split by x. One endpoint of the shortest edge of τ is p. Let q denote the other endpoint. Because p is an input vertex, q is also an input vertex. This is because p did not appear in $\mathcal{V}$ earlier than q by the definition of parent-child relation, which can only happen if q is also an input vertex. This means that the empty ball centered at x with radius r_x has two input vertices, namely p and q, on its boundary. Therefore, $r_x = \|p-x\| \geq f(x)$.

Consider the case where p is not an input vertex. If x has type 3, then r_p is at most the shortest edge length of the tetrahedron split by x. Then $r_x = X \geq \rho_0 r_p$. Suppose that x has type 1 or 2. If p has type 1, or p has type 2 and x has type 2, then $\|p-x\| \geq f(x)$ by Lemma 4.1 and $r_x \geq \|p-x\|/\sqrt{2}$ by Lemma 4.2. Then $r_x \geq f(x)/\sqrt{2}$. The remaining cases are that p has type 2 and x has type 1, or p has type 3.

Case 1: p lies inside $\widehat{x}$. Thus p was rejected by QUALMESH and $r_x = X$. Let τ be the subsegment or subfacet of which $\widehat{x}$ is the smallest circumsphere.

Case 1.1: τ is a subsegment ab. Let a be the vertex of τ nearest to p. Then $\angle pxa \leq \pi/2$. It follows that $\|p-a\| \leq \sqrt{2}X = \sqrt{2}r_x$. The vertex a was in $\mathcal{V}$ when p was rejected. Thus $r_p \leq \|p-a\|$. Hence, $r_x \geq r_p/\sqrt{2}$.

Case 1.2: τ is a subfacet abc. The Voronoi diagram of a, b, and c divides abc into three regions, each owned by a vertex of abc. Rule 2 enforces that τ contain the orthogonal projection of p. Thus we can assume that the projection of p lies in the region, say, owned by a. Let H be the plane that is perpendicular to abc and passes through a and p. Let y be the center of the circle $H \cap \widehat{x}$. As the projection of p lies inside the region owned by a, $\angle pya \leq \pi/2$. This implies that $\|p-a\|^2 \leq \|y-p\|^2 + \|y-a\|^2 \leq \|y-p\|^2 + X^2$. As p lies inside $\widehat{x} \cap H$, $\|y-p\| < \text{radius}(H \cap \widehat{x}) \leq X$. It follows that $\|p-a\| \leq \sqrt{2}X = \sqrt{2}r_x$. The vertex a was in $\mathcal{V}$ when p was rejected. Thus $r_p \leq \|p-a\|$. Hence, $r_x \geq r_p/\sqrt{2}$.

Case 2: p does not lie inside $\widehat{x}$. We have $\|p-x\| \geq r_p/\sqrt{2}$ by Lemma 4.1(ii) and $r_x \geq \|p-x\|/\sqrt{2}$ by Lemma 4.2. Thus $r_x \geq r_p/2$. □

5. Vertex-to-vertex distances. In this section, we apply Lemma 4.3 to prove lower bounds on the insertion radii and intervertex distances. In the process, we also prove that the vertex gap property holds throughout the algorithm. We will use these results in section 7 to prove several guarantees provided by weighted Delaunay refinement. We need the following relation involving local feature sizes and insertion radii.

LEMMA 5.1. *Let x be a vertex with parent p. If $r_x \geq c \cdot r_p$, then $f(x)/r_x \leq f(p)/(c \cdot r_p) + \sqrt{2}$.*

Proof. Recall that when x is inserted, x is the center of the smallest circumsphere of a subsegment, subfacet, or tetrahedron. Let $\widehat{x}$ denote this circumsphere. If x has type 3, then $r_x = \|p-x\|$. If x has type 1 or type 2, then $r_x \geq \|p-x\|/\sqrt{2}$ by Lemma 4.2. Starting with the Lipschitz condition, we get

$$\begin{aligned} f(x) &\leq f(p) + \|p-x\| \\ &\leq \frac{f(p)}{c \cdot r_p} \cdot r_x + \sqrt{2}r_x, \end{aligned}$$

which implies that $f(x)/r_x \leq f(p)/(c \cdot r_p) + \sqrt{2}$. □

The following are the constants of proportionality in Lemma 5.2, the main result in this subsection:

$$C_1 = \frac{7\sqrt{2}\rho_0}{\rho_0 - 4}, \qquad C_2 = \frac{3\sqrt{2}\rho_0 + 2\sqrt{2}}{\rho_0 - 4}, \qquad C_3 = \frac{\sqrt{2}\rho_0 + 3\sqrt{2}}{\rho_0 - 4}, \qquad \omega_0 = \frac{1}{2(1+C_1)}.$$

Note that whenever $\rho_0 > 4$, we have $C_1 > C_2 > C_3 > \sqrt{2}$.

LEMMA 5.2. *Let x be a vertex of $\mathcal{P}$ or a vertex inserted or rejected by* QUALMESH. *We have the following invariants for $\rho_0 > 4$:*

(i) *If x is a vertex of $\mathcal{P}$ or the parent of x is a vertex of $\mathcal{P}$, then $r_x \geq f(x)/\sqrt{2} > f(x)/C_3$. Otherwise, if x has type i for $1 \leq i \leq 3$, then $r_x \geq f(x)/C_i$.*

(ii) *For any other vertex y that appears in $\mathcal{V}$ currently, $\|x-y\|$ is greater than or equal to $\max\{f(x)/C_1, f(y)/(1+C_1)\}$.*

(iii) *If x is inserted by* QUALMESH, *the vertex gap property holds afterwards.*

Proof. We prove by induction. Invariant (i) holds before QUALMESH starts (the basis case). Clearly, invariant (i) is not affected by pumping a vertex. Thus it suffices to prove invariant (i) where x is inserted or rejected by QUALMESH. Let p be the

parent of x. If p is an input vertex, Lemma 4.3 implies that $r_x \geq f(x)/\sqrt{2}$. Suppose that p is not an input vertex. We assume inductively that invariant (i) holds for p and we conduct a case analysis. If x has type 3, then $r_x \geq \rho_0 \cdot r_p$ by Lemma 4.3. By induction, we get $f(p) \leq C_1 r_p$ regardless of the type of p. By Lemma 5.1, we get

$$\frac{f(x)}{r_x} \leq \frac{C_1}{\rho_0} + \sqrt{2} = C_3.$$

If x has type 2, the proof of Lemma 4.3 reveals that $r_x \geq f(x)/\sqrt{2} > f(x)/C_2$ when p has type 1 or 2. When p has type 3, Cases 1 and 2 in the proof of Lemma 4.3 reveal that $r_x \geq r_p/2$. By the induction assumption, $f(p) \leq C_3 r_p$. Then, by Lemma 5.1, we get

$$\frac{f(x)}{r_x} \leq 2C_3 + \sqrt{2} = C_2.$$

If x has type 1, then the proof of Lemma 4.3 reveals that $r_x \geq f(x)/\sqrt{2} > f(x)/C_1$ when p has type 1. When p has type 2 or 3, Cases 1 and 2 in the proof of Lemma 4.3 reveal that $r_x \geq r_p/2$. By the induction assumption, $f(p) \leq C_2 r_p$. Then, by Lemma 5.1, we get

$$\frac{f(x)}{r_x} \leq 2C_2 + \sqrt{2} = C_1.$$

This proves that invariant (i) holds in general.

Consider invariant (ii). For any vertex y that appears in $\mathcal{V}$ currently, $\|x - y\| \geq r_x \geq f(x)/C_1$. Because of $f(x) \geq f(y) - \|x - y\|$, we also get $\|x - y\| \geq f(x)/C_1 \geq f(y)/C_1 - \|x - y\|/C_1$. It follows that $\|x - y\| \geq f(y)/(1 + C_1)$.

Consider invariant (iii). It follows from invariant (ii) that for any vertices u and v in $\mathcal{V}$, $\|u - v\| \geq f(u)/(1 + C_1)$. By our choice of values of C_1 and ω_0, $f(u)/(1 + C_1) = 2\omega_0 f(u)$. This proves that the vertex gap property still holds. □

6. Effect of pumping. Let $\mathcal{V}$ denote a set of unweighted points. Let $\operatorname{Conv}\mathcal{V}$ denote the convex hull of $\mathcal{V}$. Let $\widehat{\mathcal{V}}$ be the weighted points obtained after some weight assignment to points in $\mathcal{V}$. We have already defined $N(x)$ to be the distance to the nearest neighbor in $\mathcal{V}$ for any $x \in \mathcal{V}$. We extend the definition for any $x \in \mathbb{R}^3$ by letting $N(x)$ denote the Euclidean distance from x to its *second* nearest neighbor in $\mathcal{V}$ for any point $x \in \mathbb{R}^3$. If $x \in \mathcal{V}$, then $N(x)$ still denotes the nearest neighbor distance of x. We say $\widehat{\mathcal{V}}$ has *weight property* $[\omega]$ for some $\omega \in (0, 1/2)$ if $U \leq \omega N(u)$ for each $\widehat{u} \in \widehat{\mathcal{V}}$. Let $\operatorname{Del}\widehat{\mathcal{V}}$ denote the weighted Delaunay triangulation of $\widehat{\mathcal{V}}$. $\operatorname{Del}\widehat{\mathcal{V}}$ has *ratio property* $[\rho]$ if the orthoradius-edge ratio of every tetrahedron in $\operatorname{Del}\widehat{\mathcal{V}}$ is at most ρ.

The work of Cheng et al. [7] suggests that $\operatorname{Del}\mathcal{V}$ and $\operatorname{Del}\widehat{\mathcal{V}}$ behave similarly given the ratio and weight properties, as follows.

LEMMA 6.1 (Claim 7 in [7]). *Let $\mathcal{V}$ be a periodic point set. If $\operatorname{Del}\mathcal{V}$ has ratio property $[\rho]$ and $\widehat{\mathcal{V}}$ has weight property $[\omega]$, then $\operatorname{Del}\widehat{\mathcal{V}}$ has ratio property $[\rho']$ for some ρ' depending on ρ and ω.*

In this section, we prove a version of the Lemma 6.1 for dealing with a finite point set (see Lemma 6.6). There are two differences from Lemma 6.1. First, $\mathcal{V}$ is a finite point set instead of a periodic point set. Second, we need an extra condition that the orthocenter of each tetrahedron in $\operatorname{Del}\widehat{\mathcal{V}}$ lie inside $\operatorname{Conv}\mathcal{V}$. Then the rest of Lemma 6.1 carries over.

We need three results from Talmor's thesis [16]. We state them below and include the proof of Lemma 6.3, as we need an inequality in the proof later. For a point $p \in \mathcal{V}$, let $V_p(\mathcal{V})$ denote the Voronoi cell owned by p in the Voronoi diagram of $\mathcal{V}$. Let b_p and B_p be balls centered at p such that $\text{radius}(b_p) = N(p)/2$ and $\text{radius}(B_p) = \rho L \cdot N(p)$, where L is a constant used in the next lemma.

LEMMA 6.2 (Lemma 3.4.3 in [16]). *If* $\text{Del}\,\mathcal{V}$ *has ratio property* $[\rho]$*, the lengths of two adjacent edges of* $\text{Del}\,\mathcal{V}$ *differ by at most some constant factor* L *depending on* ρ*.*

LEMMA 6.3 (Lemma 3.5.1 in [16]). *Assume that* $\text{Del}\,\mathcal{V}$ *has ratio property* $[\rho]$*. For each* $p \in \mathcal{V}$*,* $V_p(\mathcal{V})$ *contains* b_p*, and* B_p *contains all vertices of* $V_p(\mathcal{V})$*.*

Proof. It is obvious that $b_p \subseteq V_p(\mathcal{V})$ as $\text{radius}(b_p) = N(p)/2$. Let $v \in \mathcal{V}$ such that $\|p - v\| = N(p)$. Then pv is an edge of $\text{Del}\,\mathcal{V}$. Let τ be some tetrahedron in $\text{Del}\,\mathcal{V}$ incident to p. Let pq be an edge of τ. Using Lemma 6.2, we get

$$\|p - q\| \leq L \cdot \|p - v\| = L \cdot N(p). \tag{6.1}$$

Let z be the circumcenter of τ; i.e., z is a vertex of $V_p(\mathcal{V})$. The ratio property implies that

$$\|p - z\| \leq \rho \cdot \|p - q\| \leq \rho L \cdot N(p) = \text{radius}(B_p).$$

Thus, B_p contains all vertices of $V_p(\mathcal{V})$. □

LEMMA 6.4 (Theorem 3.6.2 in [16]). *Assume that* $\text{Del}\,\mathcal{V}$ *has ratio property* $[\rho]$*. Let* xz *be a line segment lying inside* $\bigcup_{p\in\mathcal{V}} V_p(\mathcal{V}) \cap B_p$*. Let* $\widehat{z}$ *be the sphere centered at* z *with radius* $\|x - z\|$*. Then there is a constant* $C > 0$ *such that if* $\widehat{z}$ *is empty, then* $N(z) \leq C \cdot N(x)$*.*

Next, we apply Lemma 6.3 to show that B_p is so large that $V_p(\mathcal{V}) \cap B_p$ contains $V_p(\mathcal{V}) \cap \text{Conv}\,\mathcal{V}$. The implication is that $\bigcup_{p\in\mathcal{V}} V_p(\mathcal{V}) \cap B_p$ contains the Voronoi diagram of $\mathcal{V}$ clipped within $\text{Conv}\,\mathcal{V}$.

LEMMA 6.5. *Assume that* $\text{Del}\,\mathcal{V}$ *has ratio property* $[\rho]$*. For each* $p \in \mathcal{V}$*,* $V_p(\mathcal{V}) \cap \text{Conv}\,\mathcal{V} \subseteq V_p(\mathcal{V}) \cap B_p$*.*

Proof. We first prove that $V_p(\mathcal{V}) \cap \text{Conv}\,\mathcal{V} \subseteq B_p$. If $V_p(\mathcal{V})$ is bounded, then $V_p(\mathcal{V}) \subseteq B_p$ by Lemma 6.3. It follows that $V_p(\mathcal{V}) \cap \text{Conv}\,\mathcal{V} \subseteq B_p$. Suppose that $V_p(\mathcal{V})$ is unbounded. Then p is an extreme vertex of $\text{Conv}\,\mathcal{V}$. Let T be the set of boundary triangles of $\text{Conv}\,\mathcal{V}$ incident to p. For each triangle $t \in T$, let H_t denote the supporting plane of t, and let H_t^+ denote the halfspace bounded by H_t that contains $\text{Conv}\,\mathcal{V}$. By convexity, $V_p(\mathcal{V}) \cap \text{Conv}\,\mathcal{V} \subseteq V_p(\mathcal{V}) \cap \bigcap_{t\in T} H_t^+$. We show that B_p contains the vertices of $V_p(\mathcal{V}) \cap \bigcap_{t\in T} H_t^+$. A vertex z of $V_p(\mathcal{V}) \cap \bigcap_{t\in T} H_t^+$ has one of three types:

1. z is a vertex of $V_p(\mathcal{V})$. By Lemma 6.3, $z \in B_p$.
2. z is the intersection of some edge pq in $\text{Del}\,\mathcal{V}$ with a facet of $V_p(\mathcal{V})$. By (6.1), $\|p - q\| \leq L \cdot N(p) \leq \text{radius}(B_p)$, and so $z \in B_p$.
3. z is the intersection of H_t for some $t \in T$ and some edge of $V_p(\mathcal{V})$. Let q and r be the other two vertices of t. Let $\mathcal{Q}$ be the convex quadrilateral on H_t bounded by the bisector plane of p and q, the bisector plane of p and r, pq, and pr. See Figure 6.1.

   Let u be the vertex of $\mathcal{Q}$ diagonally opposite p. Observe that z lies inside $\mathcal{Q}$. Thus we are done if we can show that $\mathcal{Q} \subseteq B_p$. By (6.1), $\|p-q\| \leq \text{radius}(B_p)$ and $\|p - r\| \leq \text{radius}(B_p)$. It follows that p, q, and r lie inside B_p. Let $\theta = \angle pqr$ and $\beta = \angle prq$. The angle of $\mathcal{Q}$ at u is $\theta + \beta$. After splitting this

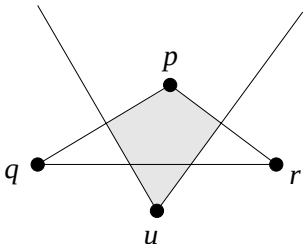


FIG. 6.1. *The shaded convex quadrilateral is* $\mathcal{Q}$.

angle into two with the diagonal pu, let γ denote the one on the same side as pq. Without loss of generality, assume that $\gamma \geq (\theta + \beta)/2$, which is at least $\min\{\theta, \beta\}$. By the ratio property, $\rho \geq \min\{1/(2\sin\theta), 1/(2\sin\beta)\}$. It follows that $\rho \geq 1/(2\sin\gamma)$. We have

$$\begin{aligned} \|p-u\| &= \frac{\|p-q\|}{2\cdot\sin\gamma} \\ &\leq \rho\cdot\|p-q\| \\ &\overset{(6.1)}{\leq} \rho L\cdot N(p) \\ &= \text{radius}(B_p). \end{aligned}$$

Therefore, $\mathcal{Q} \subseteq B_p$, and hence z lies inside B_p.

Note that $V_p(\mathcal{V}) \cap \bigcap_{t\in T} H_t^+$ is bounded. We conclude that $V_p(\mathcal{V}) \cap \operatorname{Conv}\mathcal{V} \subseteq V_p(\mathcal{V}) \cap \bigcap_{t\in T} H_t^+ \subseteq B_p$. Finally, $V_p(\mathcal{V}) \cap \operatorname{Conv}\mathcal{V} = V_p(\mathcal{V}) \cap V_p(\mathcal{V}) \cap \operatorname{Conv}\mathcal{V} \subseteq V_p(\mathcal{V}) \cap B_p$. ∎

Finally, we apply Lemmas 6.4 and 6.5 to prove the main result of this subsection.

LEMMA 6.6. *Let* $\mathcal{V}$ *be a finite point set. Assume that* $\operatorname{Del}\mathcal{V}$ *has ratio property* $[\rho]$, $\widehat{\mathcal{V}}$ *has weight property* $[\omega]$, *and the orthocenter of each tetrahedron in* $\operatorname{Del}\widehat{\mathcal{V}}$ *lies inside* $\operatorname{Conv}\mathcal{V}$. *Then* $\operatorname{Del}\widehat{\mathcal{V}}$ *has ratio property* $[\rho']$ *for some constant* ρ' *depending on* ρ *and* ω.

Proof. Let $\widehat{z}$ be the orthosphere of a tetrahedron τ in $\operatorname{Del}\widehat{\mathcal{V}}$. That is, z is the orthocenter of τ. Let qr be the shortest edge of τ. Let x be the intersection point $qz \cap \widehat{z}$.

By assumption, z lies inside $\operatorname{Conv}\mathcal{V}$. Thus we have $xz \subseteq \operatorname{Conv}\mathcal{V}$ by convexity. Using the fact that $\bigcup_{p\in\mathcal{V}} V_p(\mathcal{V}) = \mathbb{R}^3$, we get $xz \subseteq \operatorname{Conv}\mathcal{V} \cap \bigcup_{p\in\mathcal{V}} V_p(\mathcal{V}) = \bigcup_{p\in\mathcal{V}} V_p(\mathcal{V}) \cap \operatorname{Conv}\mathcal{V}$. Lemma 6.5 further implies that $xz \subseteq \bigcup_{p\in\mathcal{V}} V_p(\mathcal{V}) \cap B_p$. As $\widehat{z}$ is empty, we can apply Lemma 6.4 to xz and $\mathcal{V}$. We get

$$Z \leq N(z) \leq C\cdot N(x). \tag{6.2}$$

By the Lipschitz property, $N(x) \leq N(q) + \|q-x\|$. Because $\widehat{q}$ and $\widehat{z}$ intersect and $x = qz \cap \widehat{z}$, x lies inside $\widehat{q}$. By the weight property, the radius of $\widehat{q}$ is at most $\omega N(q)$, and so $\|q-x\| \leq \omega N(q)$. Thus, we have $N(x) \leq (1+\omega)N(q)$. As q and r are vertices in $\mathcal{V}$, $N(q) \leq \|q-r\|$. It follows that

$$N(x) \leq (1+\omega)\cdot\|q-r\|. \tag{6.3}$$

Substituting (6.3) into (6.2), we get $Z \leq C\cdot(1+\omega)\cdot\|q-r\|$. Recall that qr is the shortest edge of τ. Hence, ρ' can be set to be $C\cdot(1+\omega)$. ∎

7. Guarantees. We establish four guarantees for QualMesh. The first two guarantees, termination and gradedness, follow from Lemma 5.2. The absence of sliver and the size optimality are guaranteed using Lemma 5.2, Lemma 6.6, and some other results to be proved.

7.1. Termination and gradedness.

Theorem 7.1. QualMesh *terminates with a graded mesh.*

Proof. It follows from Lemma 5.2(ii) that any two vertices u and v at any stage of the algorithm must satisfy $\|u - v\| \geq f(u)/(1 + C_1) \geq f_{min}/(1 + C_1)$, where f_{min} is the minimum local feature size in the domain. If we center disjoint balls of radii $f_{min}/(2 + 2C_1)$ at the mesh vertices, then we can pack at most $24(1 + C_1)^3 \text{volume}(D)/(4\pi f_{min}^3)$ such balls inside a bounded domain D. Thus the algorithm must terminate. Gradedness follows from the vertex gap property. □

7.2. Conformity. After pumping in step 3, is $\text{Del}\,\widehat{\mathcal{V}}$ conforming? The vertices in $\widehat{\mathcal{V}}$ partition the input segments into weighted-subsegments. For any input facet F, the two-dimensional weighted Delaunay triangulation of vertices on F partition F into weighted-subfacets. We show that $\text{Del}\,\widehat{\mathcal{V}}$ is conforming by showing that no weighted-subsegment or weighted-subfacet is encroached.

Theorem 7.2. *No weighted-subsegment or weighted-subfacet is encroached upon during the completion of* QualMesh.

Proof. Assume to the contrary that a vertex $\widehat{v}$ in $\text{Del}\,\widehat{\mathcal{V}}$ encroaches upon a weighted-subsegment or weighted-subfacet τ.

Consider the case in which τ is a weighted-subsegment. Observe that the smallest circumsphere C of τ encloses the smallest orthosphere O of τ. Thus, the bisector H of C and O avoids C, and so H avoids τ too. τ lies in the halfspace H^+ bounded by H such that $\pi(p, C) \leq \pi(p, O)$ for all points $p \in H^+$. Observe that in order that $\widehat{v}$ encroach upon τ, τ contains the orthogonal projection of v onto the segment containing τ. Thus, $v \in H^+$, and so $\pi(\widehat{v}, C) \leq \pi(\widehat{v}, O) < 0$. The weight of $\widehat{v}$ is at most the weight used for v in Rule 4. However, this contradicts the fact that v, with that weight, did not encroach upon τ.

Consider the case in which τ is a weighted-subfacet on a facet F. By Lemma 3.6, we can assume that τ contains the projection of v. Let O be the smallest orthosphere of τ. The two-dimensional unweighted Delaunay triangulation of vertices in $\mathcal{V}$ on F partitions F into subfacets. Let τ' be the subfacet that contains the projection of v on τ. Let C be the smallest circumsphere of τ'. We claim that the bisector plane H of C and O avoids τ'. Otherwise, H intersects C, which implies that C and O intersect (H passes through their intersection). Because τ' has vertices on both sides of H, a vertex of τ' must lie inside O, a contradiction. Observe that τ' lies in the halfspace H^+ bounded by H such that $\pi(p, C) \leq \pi(p, O)$ for all point $p \in H^+$. Thus, $\pi(\widehat{v}, C) \leq \pi(\widehat{v}, O) < 0$. The weight of $\widehat{v}$ is at most the weight used for v in Rule 4. However, this contradicts the fact that v, with that weight, did not encroach upon τ'. □

7.3. No sliver. Slivers incident to an unweighted vertex p are eliminated by pumping p. In sliver exudation, p is pumped within the weight interval $[0, \omega_0^2 N(p)^2]$. During the pumping, tetrahedra incident to p change at discrete instances. For pumping to work, two conditions (Lemmas 7.3 and 7.4) must hold over the entire interval of pumping. First, the lengths of edges incident to p are within a constant factor. Second, only a constant number of tetrahedra can be incident to p. These two conditions are proved by Cheng et al. [7] for periodic point set using Lemma 6.1. Lemma 6.6 is

the analogue of Lemma 6.1 for finite point set. By Theorem 7.2 and Lemma 3.3(iii), the conditions of Lemma 6.6 are satisfied. Thus we can prove the two conditions for finite point set using exactly the same proofs as those used in [7]. For completeness, we sketch the proofs below. Let $K(\mathcal{V})$ be the graph consisting of edges in $\operatorname{Del}\widehat{\mathcal{V}}$ for all $\widehat{\mathcal{V}}$ with weight property $[\omega_0]$.

LEMMA 7.3 (Claim 10 in [7]). *Assume that* $\operatorname{Del}\mathcal{V}$ *has ratio property* $[\rho_0]$. *The lengths of any two adjacent edges in* $K(\mathcal{V})$ *are within a constant factor* $\nu_0 \geq 1$ *depending on* ρ_0 *and* ω_0.

Proof (sketch). Let p be a vertex in $\mathcal{V}$. First, consider a triangle $pqu \in \operatorname{Del}\widehat{\mathcal{V}}$ for some $\widehat{\mathcal{V}}$ with weight property $[\omega_0]$. Let Z be the radius of the smallest orthosphere of pqu. By Lemma 6.6, $Z \leq \rho' \cdot \|p-q\|$ and $Z \leq \rho' \cdot \|p-u\|$. On other hand, by the weight property, a constant fraction of pq lies outside $\widehat{p}$ and $\widehat{q}$. This fraction of pq lies inside the smallest orthosphere of pqu, and so $Z = \Omega(\|p-q\|)$. Similarly, $Z = \Omega(\|p-u\|)$. It follows that $\|p-q\|$ and $\|p-u\|$ differ by at most some constant factor k_1.

Second, consider the case in which pq and pu are two edges in $K(\mathcal{V})$ such that $\angle qpu$ is less than some constant angle bound η. Assume that $\widehat{\mathcal{V}_1}$ and $\widehat{\mathcal{V}_2}$ denote the two weighted versions of $\mathcal{V}$ such that $pq \in \operatorname{Del}\widehat{\mathcal{V}_1}$ and $pu \in \operatorname{Del}\widehat{\mathcal{V}_2}$. Let H be the plane passing through pqu. We pick the orthosphere $\widehat{z}$ of a tetrahedron τ in $\operatorname{Del}\widehat{\mathcal{V}_1}$ that is incident on pq. We intersect $\widehat{z}$ with H to obtain a circle $\widehat{y}$ centered at y with radius Y. Note that $\widehat{y}$ is orthogonal to the circles $\widehat{p} \cap H$ and $\widehat{q} \cap H$. By Lemma 6.6, the radius of $\widehat{z}$ is at most $\rho' \cdot \|p-q\|$. Clearly, Y is at most the radius of $\widehat{z}$. So $Y \leq \rho' \cdot \|p-q\|$. This implies that pq cuts deeply into $\widehat{y}$. As u lies outside $\widehat{y}$, for sufficiently small η (dependent on ρ' and ω_0), pu cannot be much shorter than pq. By symmetry, pq cannot be much shorter than pu. Thus, $\|p-q\|$ and $\|p-u\|$ differ by at most some constant factor k_2.

Next, we deal with all incident edges of p. Let S be a unit sphere centered at p. We take a maximal packing of disjoint spherical caps with angular radii $\eta/4$ on S. The number of such caps is a constant m dependent on η. Then we expand the angular radius of each cap to $\eta/2$. The expanded caps cover S. Each incident edge pq projects radially to a vertex q' on S. Each triangle pqr in $\operatorname{Del}\widehat{\mathcal{V}}$ for some $\widehat{V}$ with weight property $[\omega_0]$ projects radially to an arc $q'r'$ on S. This yields a connected graph embedded on S. Suppose that we walk from q' to an arbitrary vertex u' within the graph. If the walk stays within a cap, by our second observation, the edge length increases by at most a factor of k_2. If the walk enters a new cap, by our first observation, the edge length increases by at most a factor of k_1. If the walk returns to a cap visited before, the whole detour increases the edge length by at most a factor of k_2. As there are m caps, we conclude that $\|p-q\|$ and $\|p-u\|$ differ by at most a factor of $k_2^m k_1^{m-1}$. □

LEMMA 7.4 (Claim 11 in [7]). *Assume that* $\operatorname{Del}\mathcal{V}$ *has ratio property* $[\rho_0]$. *The degree of every vertex in* $K(\mathcal{V})$ *is bounded by some constant* δ_0 *depending on* ρ_0 *and* ω_0.

Proof (sketch). Let p be a vertex in $\mathcal{V}$. Let L be the length of the longest incident edge of p in $K(\mathcal{V})$. Let r be a neighbor of p in $K(\mathcal{V})$. By Lemma 7.3, $\|p-r\| \geq L/\nu_0$. The nearest neighbor of r is a Delaunay neighbor of r. Thus, by Lemma 7.3 again, the nearest neighbor distance of r is at least L/ν_0^2. It follows that, at the neighbors of p, we can center disjoint balls of radius $L/(2\nu_0^2)$. Observe that all these balls lie inside the ball centered at p with radius $L + L/(2\nu_0^2)$. Thus, a packing argument shows that p has $O(1)$ neighbors. □

Cheng et al. [7] proved that a sliver incident to p can remain weighted Delaunay only within a subinterval of width $O(\sigma_0 N(p)^2)$ during pumping. As the number of

tetrahedra in $K(\mathcal{V})$ incident to p is bounded by a constant, if we choose σ_0 properly, the intervals over which slivers remain incident to a vertex p can be made small enough so that there is a subinterval over which no sliver is incident to p. This is the key idea in [7] in the proof that pumping can eliminate slivers for periodic point sets. The freedom in choosing σ_0 reveals that it is unnecessary to use exactly the weight interval $[0, \omega_0^2 N(p)^2]$. It is equally good that the weight interval contains $[0, \overline{\omega}_0^2 N(p)^2]$ for some constant $\overline{\omega}_0 \leq \omega_0$.

We would like to employ Lemmas 7.3 and 7.4 with $\mathcal{V}$ being a finite point set. To this end, we first need to guarantee that the pumping in step 3 of QualMesh uses a weight interval $[0, \overline{\omega}_0^2 N(p)^2]$ and that the resulting weighted point set $\widehat{\mathcal{V}}$ has weight property $[\omega_0]$ for some constants $\overline{\omega}_0 \leq \omega_0$. The weight property $[\omega_0]$ follows from the vertex gap property. Lemma 7.5 tells us how to set $\overline{\omega}_0$.

Lemma 7.5. *Let $\mathcal{M}$ be the mesh obtained at the end of step 2 of* QualMesh. *For any vertex v in $\mathcal{M}$, its nearest neighbor distance is at most $2\sqrt{2}f(v)$.*

Proof. Let B_v denote the ball centered at v of radius $N(v)/2$. Assume to the contrary that $2\sqrt{2}f(v) < N(v)$. Then B_v intersects two disjoint elements of $\mathcal{P}$. B_v cannot contain any vertex in $\mathcal{V}$. So B_v intersects the interior of subsegments or subfacets. Let E be the nearest subsegment or subfacet to v that B_v intersects. Let $\tilde{v}$ be the orthogonal projection of v onto the affine hull of E. Note that $\tilde{v}$ lies on E. The Voronoi diagram of the vertices of E partitions E into regions, each owned by one vertex of E. Assume that $\tilde{v}$ lies in the region owned by the vertex w of E. Because $\|v - w\| \geq \text{radius}(B_v) > \sqrt{2}f(v)$ and $\|v - \tilde{v}\| \leq f(v)$, we have

$$\|v - \tilde{v}\| < \|v - w\|/\sqrt{2}. \tag{7.1}$$

Let $\widehat{x}$ be the smallest circumsphere of E. We have $\|v - x\|^2 = \|\tilde{v} - x\|^2 + \|v - \tilde{v}\|^2$. Because $\tilde{v}$ lies in the region owned by w, $\angle x\tilde{v}w \geq \pi/2$, which implies that $\|\tilde{v} - x\|^2 \leq \|w - x\|^2 - \|\tilde{v} - w\|^2$. Therefore,

$$\begin{aligned} \|v - x\|^2 &\leq \|w - x\|^2 - \|\tilde{v} - w\|^2 + \|v - \tilde{v}\|^2 \\ &= \|w - x\|^2 - \|v - w\|^2 + 2\|v - \tilde{v}\|^2 \\ &\overset{(7.1)}{<} \|w - x\|^2. \end{aligned}$$

However, v lies inside $\widehat{x}$ then, and so v encroaches upon E, a contradiction. □

In step 3 of QualMesh, we pump p using the weight interval $[0, \omega_0^2 f(p)^2]$. This interval contains $[0, \omega_0^2 N(p)^2/8]$ by Lemma 7.5. Using Lemmas 7.3 and 7.4, the same proof in [7] shows that pumping eliminates slivers for the finite point set $\mathcal{V}$. For completeness, we sketch the proof below. Algorithmically, we can use flips to generate new tetrahedra as pumping progresses and stop when no sliver is incident to p.

Theorem 7.6. *There is a constant $\sigma_0 > 0$ such that $\sigma(\tau) > \sigma_0$ for every tetrahedron τ in the output mesh of* QualMesh.

Proof (sketch). Let $pqrs$ be a sliver in some $\widehat{\mathcal{V}}$ with weight property $[\omega_0]$. We are to analyze what happens to $pqrs$ when p is pumped with weight from the interval $[0, \omega_0^2 f(p)^2]$. Let W_{qrs} be the subinterval such that $pqrs$ may remain weighted Delaunay when $P^2 \in W_{qrs}$.

We claim that $|W_{qrs}| = O(\sigma_0 N(p)^2)$. Let L be the shortest edge length of $pqrs$. Let $\widehat{z}$ be the orthosphere of $pqrs$. Let $H(P)$ be the signed distance of z from the plane passing through qrs when p has weight P^2. $H(P)$ is positive if p and z lie on the same side and $H(P)$ is negative otherwise. Observe that $Z^2 = H(P)^2 + Y^2$, where Y is the radius of smallest orthosphere of qrs. By Lemma 6.6, $Z \leq \rho' L$. The

circumradius of qrs is at least $L/2$. Also, by Claim 4 in [7], the circumradius of qrs is at most $Y/\sqrt{1-4\omega_0^2}$. It follows that $H(P)^2 = Z^2 - Y^2 = O(L^2)$ or $H(P) \in [-kL, kL]$ for some constant k. By Claim 13 in [7], $H(P) = H(0) - P^2/(2D)$, where D is the distance of p from the plane passing through qrs. Substituting into $H(P) \in [-kL, kL]$ and rearranging terms, we get $2D \cdot H(0) - 2kDL \le P^2 \le 2D \cdot H(0) + 2kDL$. Thus, $|W_{qrs}| \le 4kDL$. Note that volume$(pqrs) = \Theta(L^2 D)$ and volume$(pqrs)/L^3 \le \sigma_0$ as $pqrs$ is a sliver. This implies that $D = O(\sigma_0 L)$ and so $|W_{qrs}| = O(\sigma_0 L^2)$. The nearest neighbor of p is the Delaunay neighbor of p. Thus by applying Lemma 7.3 to p and then to q, r, and s, we conclude that $L = \Theta(N(p))$. Hence, $|W_{qrs}| = O(\sigma_0 N(p)^2)$.

Finally, by Lemma 7.4, there are at most δ_0^3 slivers incident to p throughout the entire pumping. Hence there are at most δ_0^3 forbidden subintervals. Their total length is at most $k'\sigma_0\delta_0^3 N(p)^2$ for some constant k'. By Lemma 7.5, the weight interval $[0, \omega_0^2 f(p)^2]$ contains $[0, \omega_0^2 N(p)^2/8]$. It follows that if $\sigma_0 < \omega_0^2/(8k'\delta_0^3)$, p can be assigned a weight within $[0, \omega_0^2 f(p)^2]$ such that p is not incident to any sliver. □

7.4. Size optimality. We prove that the size of our Delaunay mesh is within a constant factor of the size of any mesh that has bounded aspect ratio. Our proof is a combination of ideas in Ruppert's proof for the two-dimensional case [19] and ideas in Mitchell and Vavasis's proof for their octtree algorithm in higher dimensions [18].

Let T be a triangulation of the input domain that conforms to $\mathcal{P}$ and has bounded aspect ratio. Let τ be a tetrahedron in T. Denote the minimum height of τ from a vertex by $h(\tau)$. Let v_0, v_1, v_2, and v_3 be the vertices of τ. Each v_i is to be viewed as a column vector consisting of the coordinates of the vertex. Define M_τ to be the 3×3 matrix $(v_1 - v_0, v_2 - v_0, v_3 - v_0)$. Although M_τ depends on the numbering of the vertices of τ, the numbering does not affect the properties of M_τ that will be used. For a vector x, we use $\|x\|$ to denote its L_2-norm. For a square matrix A, we use $\|A\|$ to denote its spectral norm, i.e., the square root of the maximum eigenvalue of $A^t A$.

LEMMA 7.7 (Lemma 2 in [18]). *For any tetrahedron τ in T, $\|(M_\tau^{-1})^t\| = \|M_\tau^{-1}\| \le k_1/h(\tau)$ for some constant $k_1 \ge 1$.*

Proof. It is proved in [18] that $\|M_\tau^{-1}\| \le k_1/h(\tau)$ for some constant k_1. $\|A\| = \|A^t\|$ for any square matrix A. □

Between two points on two disjoint elements of T (vertices, edges, triangles, or tetrahedra), we show in the following lemma that one can always find a tetrahedron that is relatively small compared with the distance between the two points. The result is analogous to Theorem 2 in [18].

LEMMA 7.8. *Let p and q be two points in the interior of T. Let τ_p and τ_q be the tetrahedra of T containing p and q, respectively. If τ_p and τ_q do not share any vertex, then there is a tetrahedron τ in T intersecting pq such that*

(i) *$h(\tau) \le k_2\|p - q\|$ for some constant $k_2 \ge 1$;*
(ii) *τ shares a vertex with τ_q. (τ can be forced to share a vertex with τ_p instead, but τ cannot be guaranteed to share vertices with both τ_p and τ_q.)*

Proof. Define a simplicial map $\psi : T \to \mathbb{R}$ by setting $\psi(v) = 1$ for each vertex v of τ_q, and $\psi(w) = 0$ for all other vertices w. It follows that $\psi(x) = 1$ for all points $x \in \tau_q$, and $\psi(x) = 0$ for all points $x \in \tau_p$. As ψ is continuous, there exists a point u on pq such that the directional derivative of ψ at u has magnitude at least $1/\|p - q\|$. By convexity, there is a tetrahedron τ in T containing u. By the linearity of ψ on τ, $\bigtriangledown\psi$ is constant on τ. Therefore,

$$\|\bigtriangledown\psi\| \ge \frac{1}{\|p - q\|} \tag{7.2}$$

on τ. This implies that τ shares a vertex with τ_q; otherwise ψ and $\bigtriangledown\psi$ would be identically zero on τ, which is a contradiction. This proves (ii). We express $\bigtriangledown\psi$ on τ using M_τ as follows. Let v_0, v_1, v_2, v_3 be the vertices of τ. Define $r_i = \psi(v_i) - \psi(v_0)$ for $i = 1, 2, 3$. Observe that $0 \leq |r_i| \leq 1$.

CLAIM 7.1. *For any point z in τ, $\psi(z) - \psi(v_0) = (r_1, r_2, r_3)M_\tau^{-1}(z - v_0)$.*

Proof. The point z can be written as a convex combination of the vertices of τ: $z = \sum_{i=0}^{3} \lambda_i v_i$. This implies that

$$z - v_0 = \sum_{i=1}^{3} \lambda_i (v_i - v_0).$$

We view M_τ^{-1} as three row vectors $\alpha_1, \alpha_2, \alpha_3$ ordered from top to bottom. Recall that $M_\tau = (v_1 - v_0, v_2 - v_0, v_3 - v_0)$. As $M_\tau^{-1} M_\tau = I$, $\alpha_k \cdot (v_k - v_0) = 1$ and $\alpha_k \cdot (v_j - v_0) = 0$ for all $j \neq k$. It follows that

$$M_\tau^{-1}(z - v_0) = (\lambda_1, \lambda_2, \lambda_3)^t.$$

We conclude that $(r_1, r_2, r_3)M_\tau^{-1}(z - v_0) = \sum_{i=1}^{3} \lambda_i(\psi(v_i) - \psi(v_0)) = (\sum_{i=0}^{3} \lambda_i \psi(v_i)) - \psi(v_0) = \psi(z) - \psi(v_0)$. □

The claim implies that $\bigtriangledown\psi = (M_\tau^{-1})^t (r_1, r_2, r_3)^t$ on τ and so

$$\begin{aligned} \|\bigtriangledown\psi\| &\leq \|(M_\tau^{-1})^t\| \cdot \|(r_1, r_2, r_3)^t\| \\ &\leq \sqrt{3} \cdot \|(M_\tau^{-1})^t\|, \quad \text{as } 0 \leq |r_i| \leq 1 \\ &\overset{\text{Lemma 7.7}}{\leq} \sqrt{3} k_1 / h(\tau). \end{aligned}$$

Combining the above with (7.2), we obtain $h(\tau) \leq \sqrt{3} k_1 \|p - q\|$. This proves (i). □

As T has a bounded aspect ratio, it enjoys properties similar to those stated in Lemma 6.2. In particular, two tetrahedra sharing a vertex have similar minimum heights.

LEMMA 7.9 (see [17, 18]). *If two tetrahedra τ_1 and τ_2 in T share a vertex, then $h(\tau_1) \leq k_3 h(\tau_2)$ for some constant $k_3 \geq 1$.*

Next, we prove that the minimum heights of tetrahedra in T change smoothly. The result is analogous to Lemma 11 in [18].

LEMMA 7.10. *Let p and q be two points, and let τ_p and τ_q, respectively, be the two tetrahedra in T that contain them. Then $h(\tau_q) \leq k_4 \max\{h(\tau_p), \|p - q\|\}$ for some constant $k_4 \geq 1$.*

Proof. If τ_q shares a vertex with τ_p, then $h(\tau_q) \leq k_3 h(\tau_p)$, by Lemma 7.9. Consider the case where τ_q does not share a vertex with τ_p. By Lemma 7.8, there is a tetrahedron τ in T intersecting pq such that $h(\tau) \leq k_2 \|p - q\|$ and τ shares a vertex with τ_q. Starting with Lemma 7.9, we get $h(\tau_q) \leq k_3 h(\tau) \leq k_2 k_3 \|p - q\|$. □

The following lemma shows that the minimum heights of tetrahedra in T are also related to the local feature sizes. The result is analogous to Lemma 5 in [19].

LEMMA 7.11. *Let x be a point, and let τ be a tetrahedron in T containing x. Then $h(\tau) \leq k_5 f(x)$ for some constant $k_5 \geq 1$.*

Proof. Let B be the ball centered at x of radius $f(x)$. B contains two points p and q on two disjoint elements of $\mathcal{P}$. By Lemma 7.8, there is a tetrahedron τ' in T intersecting pq such that $h(\tau') \leq k_2 \|p - q\|$. Let u be a point in the intersection of τ'

and pq. By applying Lemma 7.10 to u and x, we have

$$\begin{aligned} h(\tau) &\le k_4 \max\{h(\tau'), \|u-x\|\} \\ &\le k_4 \max\{k_2\|p-q\|, \|u-x\|\} \\ &\le k_4 \max\{k_2 \cdot 2f(x), f(x)\} \\ &\le 2k_2k_4 f(x). \qquad \square \end{aligned}$$

We are now ready to prove the main theorem of this section.

THEOREM 7.12. *The output size of* QUALMESH *is within a constant factor of the size of any mesh of bounded aspect ratio for the same domain.*

Proof. Let $\mathcal{D}$ denote the domain to be meshed. Let n be the number of vertices in the Delaunay mesh output by QUALMESH. First, we show that n is at most some constant times $\int_{\mathcal{D}} \frac{dx}{f(x)^3}$. As $N(v) \ge f(v)/(1+C_1)$, we can center a ball B_v of radius $d_v = f(v)/(2+2C_1)$ at each vertex v so that all balls are disjoint. Observe that $f(x) \le f(v) + d_v$ for all $x \in B_v$. Therefore,

$$\begin{aligned} \int_{\mathcal{D}} \frac{dx}{f(x)^3} &\ge \sum_v \int_{B_v} \frac{dx}{f(x)^3} \\ &\ge \sum_v \frac{4\pi d_v^3}{3(f(v)+d_v)^3} \\ &\ge \sum_v \frac{4\pi}{3(3+2C_1)^3}. \end{aligned}$$

Therefore, we have

$$n = \sum_v 1 \le \frac{3(3+2C_1)^3}{4\pi} \int_{\mathcal{D}} \frac{dx}{f(x)^3}.$$

By Lemma 7.4, the number of tetrahedra in the Delaunay mesh is within a constant factor of n. More formally, let m be the number of tetrahedra; we have

$$m \le k_6 \int_{\mathcal{D}} \frac{dx}{f(x)^3}$$

for some constant k_6.

Next, consider a mesh T of $\mathcal{D}$ of bounded aspect ratio. For each point x in $\mathcal{D}$, define $\ell(x)$ to be the minimum height of the tetrahedra in T containing x. By Lemma 7.11, $\ell(x) \le k_5 f(x)$ and so

$$\begin{aligned} \int_{\mathcal{D}} \frac{dx}{f(x)^3} &\le k_5^3 \int_{\mathcal{D}} \frac{dx}{\ell(x)^3} \\ &\le k_5^3 \sum_{\tau \in T} \int_{\tau} \frac{dx}{h(\tau)^3} \\ &= k_5^3 \sum_{\tau \in T} \frac{\text{volume}(\tau)}{h(\tau)^3} \\ &\le k_5^3 k_7 \sum_{\tau \in T} 1, \end{aligned}$$

as $\text{volume}(\tau) \le k_7 h(\tau)^3$ for some constant k_7. Combining the above with the upper bound on m, we conclude that m is within a constant factor of the size of T. $\square$

8. Conclusions. A series of developments starting with Chew [5] and Ruppert [19] and continuing with Shewchuk [20] and Cheng et al. [7] brought the difficult problem of quality three-dimensional Delaunay meshing of bounded domains close to the solution. Li and Teng [15] recently developed a randomized point-placement strategy to generate a provably good three-dimensional Delaunay mesh of bounded domains. This paper introduces a new paradigm, weighted Delaunay refinement, which gives the first deterministic algorithm for the problem. We believe that we will add fewer points in practice because weighted Delaunay refinement uses pumping to eliminate slivers instead of point-placement.

Of course, as with previous algorithms the constants derived for the theory are miserably unsatisfactory for all practical purposes. For example, the constant $\rho_0 > 4$ is large for any practical purpose, and the constant σ_0 is extremely small. Experiments show that these constants need not be that bad in practice when sliver exudation and Delaunay refinement are used separately [13]. Will the same remain true when we combine the two into our weighted Delaunay refinement algorithm?

In QualMesh we need to compute the local feature size $f(v)$ while assigning weight to a vertex v. Although the computation of $f(v)$ is feasible, it is better if we can avoid computing it in practice. Toward this end, one can gradually increase the weight to v and check the quality of tetrahedra incident to v as they change at discrete moments. Experiments should be performed to see what bound on angles we get in practice with this strategy. We plan future work to answer these questions.

REFERENCES

[1] N. Amenta, M. Bern, and D. Eppstein, *Optimal point placement for mesh smoothing*, in Proceedings of the 8th Annual ACM–SIAM Symposium on Discrete Algorithms, New Orleans, 1997, pp. 528–537.

[2] M. Bern and P. Plassman, *Mesh generation*, in Handbook of Computational Geometry, J. Sack and J. Urrutia, eds., Elsevier, New York, 2000, pp. 291–332.

[3] M. Bern, D. Eppstein, and J. Gilbert, *Provably good mesh generation*, J. Comput. System Sci., 48 (1994), pp. 384–409.

[4] B. Chazelle, *An optimal convex hull algorithm in any fixed dimension*, Discrete Comput. Geom., 9 (1993), pp. 145–158.

[5] L. P. Chew, *Guaranteed-Quality Triangular Meshes*, Technical report TR-98-983, Computer Science Department, Cornell University, Ithaca, NY, 1989.

[6] L. P. Chew, *Guaranteed-quality Delaunay meshing in 3D*, in Proceedings of the 13th Annual ACM Symposium on Computational Geometry, Nice, France, 1997, pp. 391–393.

[7] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng, *Sliver exudation*, J. ACM, 47 (2000), pp. 883–904.

[8] T. K. Dey, C. Bajaj, and K. Sugihara, *On good triangulations in three dimensions*, Internat. J. Comput. Geom. Appl., 2 (1992), pp. 75–95.

[9] T. K. Dey and J. Pach, *Extremal problems for geometric hypergraphs*, Discrete Comput. Geom., 19 (1998), pp. 473–484.

[10] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.

[11] H. Edelsbrunner, *Geometry and Topology for Mesh Generation*, Cambridge University Press, Cambridge, England, 2001.

[12] H. Edelsbrunner, X.-Y. Li, G. L. Miller, A. Stathopoulos, D. Talmor, S.-H. Teng, A. Üngör, and N. Walkington, *Smoothing cleans up slivers*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 273–277.

[13] H. Edelsbrunner and D. Guoy, *An experimental study of sliver exudation*, in Proceedings of the 10th International Meshing Roundtable, Newport Beach, CA, 2001, pp. 307–316.

[14] P. L. George and H. Borouchaki, *Delaunay Triangulation and Meshing: Application to Finite Elements*, Hermes, Paris, 1998.

[15] X.-Y. Li and S.-H. Teng, *Generating well-shaped Delaunay meshes in 3D*, in Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, pp. 28–37.

[16] D. Talmor, *Well-Spaced Points for Numerical Methods*, Technical report CMU-CS-97-164, Department of Computer Sciences, Carnegie-Mellon University, Pittsburgh, PA, 1997.

[17] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, *Geometric separators for finite-element meshes*, SIAM J. Sci. Comput., 19 (1998), pp. 364–386.

[18] S. A. Mitchell and S. A. Vavasis, *Quality mesh generation in higher dimensions*, SIAM J. Comput., 29 (2000), pp. 1334–1370.

[19] J. Ruppert, *A Delaunay refinement algorithm for quality* 2*-dimensional mesh generation*, J. Algorithms, 18 (1995), pp. 548–585.

[20] J. R. Shewchuk, *Tetrahedral mesh generation by Delaunay refinement*, in Proceedings of the 14th Annual ACM Symposium on Computational Geometry, ACM, New York, 1998, pp. 86–95.

[21] G. Strang and G. J. Fix, *An Analysis of the Finite Element Method*, Prentice–Hall, Englewood Cliffs, NJ, 1973.

[22] S.-H. Teng and C.-W. Wong, *Unstructured mesh generation: Theory, practice and perspectives*, Internat. J. Comput. Geom. Appl., 10 (2000), pp. 227–266.

SIAM J. COMPUT.
Vol. 33, No. 1, pp. 94–136

CONFLICT-FREE COLORINGS OF SIMPLE GEOMETRIC REGIONS WITH APPLICATIONS TO FREQUENCY ASSIGNMENT IN CELLULAR NETWORKS*

GUY EVEN†, ZVI LOTKER†, DANA RON†, AND SHAKHAR SMORODINSKY‡

Abstract. Motivated by a frequency assignment problem in cellular networks, we introduce and study a new coloring problem that we call *minimum conflict-free coloring* (min-CF-coloring). In its general form, the input of the min-CF-coloring problem is a set system $(X, \mathcal{S})$, where each $S \in \mathcal{S}$ is a subset of X. The output is a coloring χ of the sets in $\mathcal{S}$ that satisfies the following constraint: for every $x \in X$ there exists a color i and a unique set $S \in \mathcal{S}$ such that $x \in S$ and $\chi(S) = i$. The goal is to minimize the number of colors used by the coloring χ.

Min-CF-coloring of general set systems is not easier than the classic graph coloring problem. However, in view of our motivation, we consider set systems induced by simple geometric regions in the plane.

In particular, we study disks (both congruent and noncongruent), axis-parallel rectangles (with a constant ratio between the smallest and largest rectangle), regular hexagons (with a constant ratio between the smallest and largest hexagon), and general congruent centrally symmetric convex regions in the plane. In all cases we have coloring algorithms that use $O(\log n)$ colors (where n is the number of regions). Tightness is demonstrated by showing that even in the case of unit disks, $\Theta(\log n)$ colors may be necessary. For rectangles and hexagons we also obtain a constant-ratio approximation algorithm when the ratio between the largest and smallest rectangle (hexagon) is a constant.

We also consider a dual problem of CF-coloring points with respect to sets. Given a set system $(X, \mathcal{S})$, the goal in the dual problem is to color the elements in X with a minimum number of colors so that every set $S \in \mathcal{S}$ contains a point whose color appears only once in S. We show that $O(\log |X|)$ colors suffice for set systems in which X is a set of points in the plane and the sets are intersections of X with scaled translations of a convex region. This result is used in proving that $O(\log n)$ colors suffice in the primal version.

Key words. conflict-free coloring, frequency assignment, approximation algorithms, computational geometry

AMS subject classifications. 68W40, 68W25, 52C45

DOI. 10.1137/S0097539702431840

1. Introduction.

Cellular networks are heterogeneous networks with two different types of nodes: base-stations (that act as servers) and clients. The base-stations are interconnected by an external fixed backbone network. Clients are connected only to base-stations; links between clients and base-stations are implemented by radio links. Fixed frequencies are assigned to base-stations to enable links to clients. Clients, on the other hand, continuously scan frequencies in search of a base-station with good reception. This scanning takes place automatically and enables smooth transitions between links when a client is mobile. Consider a client that is within the reception range of two base-stations. If these two base-stations are assigned the same frequency, then mutual interference occurs, and the links between the client and each

*Received by the editors November 1, 2002; accepted for publication (in revised form) July 23, 2003; published electronically November 14, 2003DATE. A preliminary version of this paper appeared in the Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, Vancouver, 2002, pp. 691–700.

http://www.siam.org/journals/sicomp/33-1/43184.html

†Department of Electrical Engineering Systems, Tel-Aviv University, Tel-Aviv 69978, Israel (guy@eng.tau.ac.il, zvilo@eng.tau.ac.il, danar@eng.tau.ac.il). The first and third authors were partly supported by the LSRT (Large Scale Rural Telephony) Consortium.

‡School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel (smoro@tau.ac.il).

of these conflicting base-stations are rendered too noisy to be used. A base-station may serve a client, provided that the reception is strong enough and interference from other base-stations is weak enough. The fundamental problem of frequency assignment in a cellular network is to assign frequencies to base-stations so that every client is served by some base-station. The goal is to minimize the number of assigned frequencies since spectrum is limited and costly.

We consider the following abstraction of the above problem, which we refer to as the *minimum conflict-free coloring problem (min-CF-coloring).*

DEFINITION 1.1. *Let X be a fixed domain (e.g., the plane), and let $\mathcal{S}$ be a collection of subsets of X (e.g., disks whose centers correspond to base-stations). A function $\chi : \mathcal{S} \to \mathbb{N}$ is a* CF-coloring *of $\mathcal{S}$ if, for every $x \in \bigcup_{S\in\mathcal{S}} S$, there exists a color $i \in \mathbb{N}$ such that $\{S \in \mathcal{S} : x \in S$ and $\chi(S) = i\}$ contains a single subset $S \in \mathcal{S}$.*

The goal in the min-CF-coloring problem is to find a CF-coloring that uses as few colors as possible. It is not hard to verify that, in its most general form defined above, this problem is not easier than vertex coloring in graphs and is even equally hard to approximate. An adaptation of the NP-completeness proof of minimum coloring of intersection graphs of unit disks by [CCJ90] proves that even CF-coloring of unit disks (or unit squares) in the plane is NP-complete. Since this proof is based on a reduction from coloring planar graphs, it follows that approximating the minimum number of colors required in a CF-coloring of unit disks is NP-hard for an approximation ratio of $\frac{4}{3} - \varepsilon$, for every $\varepsilon > 0$.

1.1. Our results. We restrict our attention to set systems $(X, \mathcal{R})$, where X is a set of points in the plane and $\mathcal{R}$ is a family of subsets of X that are defined by the intersections of X with closed geometric regions in the plane (e.g., disks). We refer to the members of $\mathcal{R}$ as *ranges*, and to $(X, \mathcal{R})$ as a *range-space.*

1.1.1. CF-coloring of disks. Given a finite set of disks $\mathcal{S}$, the *size-ratio* of $\mathcal{S}$ is the ratio between the largest and the smallest radiuses of disks in $\mathcal{S}$. For simplicity we assume that the smallest radius is 1. For each $i \geq 1$, let $\mathcal{S}^i$ denote the subset of disks in $\mathcal{S}$ whose radius is in the range $[2^{i-1}, 2^i)$. Let $\phi_{2^i}(\mathcal{S}^i)$ denote the maximum number of centers of disks in $\mathcal{S}^i$ that are contained in a $2^i \times 2^i$ square. We refer to $\phi_{2^i}(\mathcal{S}^i)$ as the *local density* of $\mathcal{S}^i$ (with respect to $2^i \times 2^i$ square). For a set of centers $X \subset \mathbb{R}^2$, and for any given radius r, let $\mathcal{S}_r(X)$ denote the set of (congruent) disks having radius r whose centers are the points in X.

Our main results for coloring disks are stated in the following theorem.

THEOREM 1.2.

1. *Given a finite set $\mathcal{S}$ of disks with size-ratio ρ, there exists a polynomial-time algorithm that computes a CF-coloring of $\mathcal{S}$ using $O\left(\min\left\{\sum_{i=1}^{\log(\rho)+1}(1+\log\phi_{2^i}(\mathcal{S}^i)), \log|\mathcal{S}|\right\}\right) = O\left(\min\left\{\log(\rho)\cdot\max_i\{\log\phi_{2^i}(\mathcal{S}^i)\}, \log|\mathcal{S}|\right\}\right)$ colors.*[1]
2. *Given a finite set of centers $X \subset \mathbb{R}^2$, there exists a polynomial-time algorithm that computes a coloring χ of X using $O(\log|X|)$ colors such that if we color $\mathcal{S}_r(X)$ by assigning each disk $D \in \mathcal{S}_r(X)$ the color of its center, then this is a valid CF-coloring of $\mathcal{S}_r(X)$ for every radius r.*

Tightness of Theorem 1.2 is shown by presenting, for any given integer n, a set $\mathcal{S}$ of n unit disks with $\phi_1(\mathcal{S}) = n$ for which $\Omega(\log n)$ colors are necessary in every CF-coloring of $\mathcal{S}$.

[1]For simplicity of notation, we avoid writing $\log(x+1)$ throughout the paper, even when x may equal 1, and consider $O(0)$ to be $O(1)$.

In the first part of Theorem 1.2, the disks are not necessarily congruent; that is, the size-ratio ρ may be bigger than 1. In the second part of Theorem 1.2, the disks are congruent (i.e., the size-ratio equals 1). However, the common radius is not determined in advance. Namely, the order of quantifiers in the second part of the theorem is as follows: Given the locations of the disk centers, the algorithm computes a coloring of the centers (of the disks) such that this coloring is conflict-free *for every* radius r. We refer to such a coloring as a *uniform* CF-coloring.

Uniform CF-coloring has an interesting interpretation in the context of cellular networks. Assume that base-stations are located in the disk centers X. Assume that a client located at point P has a reception range r. The client is served, provided that the disk centered at P with radius r contains a base-station that transmits in a distinct frequency among the base-stations within that disk.

Thus, uniform CF-coloring models frequency assignment in the setting of isotropic base-stations that transmit with the same power and clients with different reception ranges. Moreover, the coloring of the base-stations in a uniform CF-coloring is independent of the reception ranges of the clients.

Building on Theorem 1.2, we also obtain two bicriteria CF-coloring algorithms for disks having the same (unit) radius. In both cases we obtain colorings that use very few colors. In the first case this comes at a cost of not serving a small area that is covered by the disks (i.e., an area close to the boundary of the union of the disks). In the second case we serve the entire area, but we allow the disks to have a slightly larger radius. A formal statement of these bicriteria results follows.

THEOREM 1.3. *For every $0 < \varepsilon < 1$ and every finite set of centers $X \subset \mathbb{R}^2$, there exist polynomial-time algorithms that compute colorings as follows:*

1. *A coloring χ of $\mathcal{S}_1(X)$ using $O\left(\log \frac{1}{\varepsilon}\right)$ colors for which the following holds: The area of the set of points in $\bigcup \mathcal{S}_1(X)$ that are not served with respect to χ is at most an ε-fraction of the total area of $\mathcal{S}_1(X)$.*
2. *A coloring of $\mathcal{S}_{1+\varepsilon}(X)$ that uses $O\left(\log \frac{1}{\varepsilon}\right)$ colors such that every point in $\bigcup \mathcal{S}_1(X)$ is served.*

In other words, in the first case, the portion of the total area that is not served is an exponentially small fraction as a function of the number of colors. In the second case, the increase in the radius of the disks is exponentially small as a function of the number of colors.

1.1.2. CF-coloring of rectangles and regular hexagons. Let $\mathcal{R}$ denote a set of axis-parallel rectangles. Given a rectangle $R \in \mathcal{R}$, let $w(R)$ (respectively, $h(R)$) denote the width (respectively, height) of R. The *size-ratio* of $\mathcal{R}$ is defined by $\max\left\{\frac{w(R_1)}{w(R_2)}, \frac{h(R_1)}{h(R_2)}\right\}_{R_1,R_2\in\mathcal{R}}$.

The size-ratio of a collection of regular hexagons is simply the ratio of the longest side length to the shortest side length.

THEOREM 1.4. *Let $\mathcal{R}$ denote either a set of axis-parallel rectangles or a set of axis-parallel regular hexagons. Let ρ denote the size-ratio of $\mathcal{R}$, and let $\chi_{\mathrm{opt}}(\mathcal{R})$ denote an optimal CF-coloring of $\mathcal{R}$.*

1. *If $\mathcal{R}$ is a set of rectangles, then there exists a polynomial-time algorithm that computes a CF-coloring χ of $\mathcal{R}$ such that $|\chi(\mathcal{R})| = O((\log \rho)^2) \cdot |\chi_{\mathrm{opt}}(\mathcal{R})|$.*
2. *If $\mathcal{R}$ is a set of regular hexagons, then there exists a polynomial-time algorithm that computes a CF-coloring χ of $\mathcal{R}$ such that $|\chi(\mathcal{R})| = O(\log \rho) \cdot |\chi_{\mathrm{opt}}(\mathcal{R})|$.*

For a constant size-ratio ρ, Theorem 1.4 implies a constant-ratio approximation algorithm.

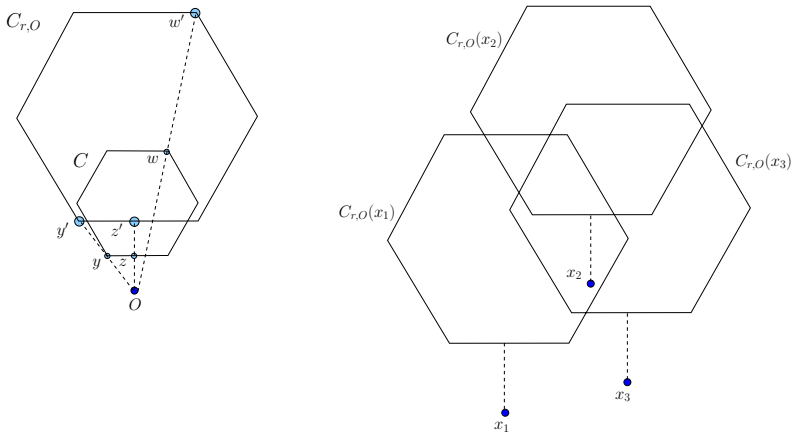


FIG. 1.1. *On the left is an example of a scaled translation $C_{r,O}$ of a regular hexagon C with respect to the point O, where the scaling factor is $r = 2$. The points y, z, and w on the small hexagon C are mapped to the points y', z', and w', respectively, on the larger hexagon $C_{r,O}$. The dashed lines correspond to the rays emanating from O toward the points y, z, and w. On the right is an additional set $X = \{x_1, x_2, x_3\}$ of three points and the corresponding set $\mathcal{C}_{r,O}(X) = \{\mathcal{C}_{r,O}(x_1), \mathcal{C}_{r,O}(x_2), \mathcal{C}_{r,O}(x_3)\}$.*

1.1.3. Uniform CF-coloring of congruent centrally symmetric convex regions. Consider a convex region C and a point O. *Scaling* by a factor $r > 0$ with respect to a center O is the transformation that maps every point $P \neq O$ to the point P' along the ray emanating from O toward P such that $|P'O| = r \cdot |PO|$. The center point O is a fixed point of the transformation of the scaling. We denote the image of C with respect to such a scaling by $C_{r,O}$. Given a point x and a scaling factor $r > 0$, we denote by $C_{r,O}(x)$ the image of $C_{r,O}$ obtained by the translation that maps O to x (see Figure 1.1). We refer to C' as a *scaled translation* of C if there exist points x, O and a scaling factor $r > 0$ such that $C' = C_{r,O}(x)$. Given a set of centers X and a scaling factor $r > 0$, the set $\mathcal{C}_{r,O}(X)$ denotes the set of scaled translations $\{C_{r,O}(x)\}_{x \in X}$.

A region $C \in \mathbb{R}^2$ is *centrally symmetric* if there exists a point O (called the *center*) such that the transformation of reflection about O is a bijection of C onto C. Note that disks, rectangles, and regular hexagons are all convex centrally symmetric regions.

The following theorem generalizes the uniform coloring result presented in part 2 of Theorem 1.2 to sets of centrally symmetric convex regions that are congruent via translations.

THEOREM 1.5. *Let C denote a centrally symmetric convex region with a center point O. Given a finite set of centers $X \subset \mathbb{R}^2$, there exists a coloring χ of X that uses $O(\log |X|)$ colors such that if we color each $c \in \mathcal{C}_{r,O}(X)$ with the color of its center, then this is a valid CF-coloring of $\mathcal{C}_{r,O}(X)$ for every scaling factor r.*

A polynomial-time constructive version of Theorem 1.5 holds when the region C is "well behaved," e.g., a disk, an ellipsoid, or a polygon. (More formally, a polynomial-time algorithm for computing Delaunay graphs of arrangements of regions $\mathcal{C}_{r,O}(X)$ is needed.)

1.2. Techniques.

1.2.1. A dual coloring problem: CF-coloring of points with respect to ranges. In order to prove Theorem 1.2, we consider the following coloring problem,

which is dual to our original coloring problem described in Definition 1.1.

DEFINITION 1.6. *Let $(X, \mathcal{R})$ denote a range-space. A function $\chi : X \to \mathbb{N}$ is a* CF-coloring *of X with respect to $\mathcal{R}$ if, for every $R \in \mathcal{R}$, there exists a color $i \in \mathbb{N}$ such that the set $\{x \in R : \chi(x) = i\}$ contains a single point.*

Note that in the original definition of CF-coloring (Definition 1.1) we were interested in coloring ranges (regions) in order to serve points contained in the ranges, while in Definition 1.6 we are interested in coloring points in order to "serve" ranges containing the points.

We give a general framework for CF-coloring points with respect to sets of ranges $\mathcal{R}$ and provide a sufficient condition under which a coloring using $O(\log |X|)$ colors can be achieved. This condition is stated in terms of a special graph constructed from $(X, \mathcal{R})$. When X is a set of points in the plane and $\mathcal{R}$ is the set of ranges obtained by intersections with disks, this graph is the standard Delaunay graph. We then study several cases in which the condition is satisfied. Theorems 1.2 and 1.5 follow by reduction to these cases. We believe that Theorem 1.7 stated below (from which Theorem 1.5 is easily derived) is of independent interest.

THEOREM 1.7. *Let C be a compact convex region in the plane, and let X be a finite set of points in the plane. Let $\mathcal{R} \subseteq 2^X$ denote the set of ranges obtained by intersecting X with all scaled translations of C. Then there exists a CF-coloring of X with respect to $\mathcal{R}$ using $O(\log |X|)$ colors.*

Recently, Pach and Tóth [PT03] proved that $\Omega(\log |X|)$ colors are required for CF-coloring *every* set X of points in the plane with respect to disks.

1.2.2. CF-coloring of chains. A *chain* $\mathcal{S}$ is a collection of subsets, each assigned a unique index in $\{1, \ldots, |\mathcal{S}|\}$, for which the following holds. For every (discrete) interval $[i, j]$, $1 \le i \le j \le |\mathcal{S}|$, there exists a point $x \in \bigcup_{S \in \mathcal{S}} S$ such that the subcollection of subsets that contains the point x equals the subcollection of subsets indexed from i to j. Moreover, for every point $x \in \bigcup_{S \in \mathcal{S}} S$, the set of indexes of subsets that contain the point x is an interval. (For an illustration, see Figure 6.2.) We show that chains of unit disks (respectively, unit squares and hexagons) are tight examples of Theorem 1.2 (respectively, Theorem 1.4); namely, every CF-coloring of a chain must use $\Omega(\log |\mathcal{S}|)$ colors, and it is possible to CF-color every chain using $O(\log |\mathcal{S}|)$ colors.

Chains also play an important role in our approximation algorithm for CF-coloring rectangles and hexagons. Loosely speaking, our coloring algorithm works by decomposing the set of rectangles into chains. An important component in our analysis is understanding and exploiting the intersections between pairs of different chains. Specifically, we show how different types of pairs of chains (see Figures 7.5 and 7.9) can "help" each other so as to go below the upper bound on the number of colors required to color chains, which is logarithmic in their size.

1.3. Related problems. As noted above, min-CF-coloring of general set systems is not easier (even to approximate) than vertex-coloring in graphs. The latter problem is of course known to be NP-hard, and is hard even to approximate [FK98]. The problem remains hard for the special case of unit disks (and squares), and it is even NP-hard to achieve an approximation ratio of $\frac{4}{3} - \varepsilon$ for every $\varepsilon > 0$ (by an adaptation of [CCJ90]).

Marathe et al. [MBH+95] studied the problem of vertex-coloring of intersection graphs of unit disks. They presented an approximation algorithm with an approximation ratio of 3. Motivated by channel assignment problems in radio networks, Krumke, Marathe, and Ravi [KMR01] presented a 2-approximation algorithm for the

distance-2 coloring problem in families of graphs that generalize intersection graphs of disks.

A natural variant of min-CF-coloring is min-CF-multicoloring. Given a collection $\mathcal{S}$ of sets, a CF-multicoloring of $\mathcal{S}$ is a mapping χ from $\mathcal{S}$ to *subsets* of colors. The requirement is that for every point $x \in \bigcup_{S\in\mathcal{S}} S$ there exists a color i such that $\{S : x \in S,\, i \in \chi(S)\}$ contains a single subset. The min-CF-multicoloring problem is related to the problem of minimizing the number of time slots required to broadcast information in a single-hop radio network. In view of this relation, it has been observed by Bar-Yehuda ([B01], based on [BGI92]) that every set system $(X, \mathcal{S})$ can be CF-multicolored using $O(\log|X| \cdot \log|\mathcal{S}|)$ colors.

Mathematical optimization techniques have been used to solve a family of frequency assignment problems that arise in wireless communication (for a comprehensive survey, see [AHK+01]). We elaborate why these frequency assignment problems do not capture min-CF-coloring. Basically, such frequency assignment problems are modeled using *interference* or *constraint* graphs. The vertices correspond to base-stations, and edges correspond to interference between pairs of base-stations. Each edge (v, w) is associated with a *penalty function* $p_{v,w} : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$, so that if v is assigned frequency $i \in \mathbb{N}$ and w is assigned frequency $j \in \mathbb{N}$, then a penalty of $p_{v,w}(i, j)$ is incurred. A typical constraint is to bound the maximum penalty on every edge. A typical cost function is the number of frequencies used. CF-coloring cannot be modeled in this fashion because CF-coloring allows for conflicts between base-stations, provided that another base-station serves the "area of conflict." Even models that use nonbinary constraints (see [DBJC98]) do not capture CF-coloring. We note that the above models take into account interferences between close frequencies, while we have ignored this issue for the sake of simplicity. We can, however, incorporate some variants of such constraints. For example, in the case of unit disks we can easily impose the constraint that, for every point x, the frequency assigned to the disk that serves x differs by at least $\delta_{\min}$ from the frequency assigned to every other disk covering x. By applying Theorem 1.2 and multiplying each color by $\delta_{\min}$, we can satisfy the above constraint while using $O\left(\min\left\{\sum_{i=1}^{\log(\rho)+1} \log \phi_{2^i}(\mathcal{S}^i), \log|\mathcal{S}|\right\} \cdot \delta_{\min}\right)$ colors (and there is an example that exhibits tightness).

Frequency assignment problems in cellular networks as well as the positioning problem of base-stations have been studied extensively; see [AKM+01, GGRV00, H01] for other models and many references. Finally, we refer to [HS03, SM03] for further work on CF-coloring problems.

Further research. Among the open problems related to our results are the following: (1) Is there a constant approximation algorithm for min-CF-coloring of unit disks and disks in general? (2) Is it possible to extend our results to min-CF-coloring with capacity constraints defined as "every base-station is given a capacity that bounds the number of clients that it can serve"?

Organization. In section 2, preliminary notions and notations are presented. In section 3 we describe our results for CF-coloring points with respect to range-spaces: We describe a general framework and several applications. In section 4 we prove our results for CF-coloring of disks (Theorems 1.2 and 1.3), which build on results from section 3. Theorem 1.5 is proved in section 5, and tightness of Theorem 1.2 is established in section 6. Our $O(1)$-approximation algorithm for rectangles is provided in section 7. In section 8 we discuss how a very similar algorithm can be applied to color regular hexagons. Finally, in section 9 we derive a couple of additional related results.

2. Preliminaries.

2.1. Combinatorial arrangements. A finite set $\mathcal{R}$ of regions (in the plane) induces the following *equivalence relation*. Every two points x, y in the plane belong to the same class if and only if they reside in exactly the same subset of regions in $\mathcal{R}$. That is, x and y are in the same equivalence class if $\{R \in \mathcal{R} : x \in R\} = \{R \in \mathcal{R} : y \in R\}$. We refer to each such equivalence class as a *cell*. The set of all cells induced by $\mathcal{R}$ is denoted by $cells(\mathcal{R})$. With a slight abuse of notation, we view the pair $(cells(\mathcal{R}), \mathcal{R})$ as a range-space. To be precise, $(cells(\mathcal{R}), \mathcal{R})$ is the following range-space: (a) the ground set is equal to a representative from every cell, and (b) the ranges are the intersections of sets in $\mathcal{R}$ with the ground set. We henceforth refer to the range-space $(cells(\mathcal{R}), \mathcal{R})$ as the *combinatorial arrangement* induced by $\mathcal{R}$; we denote this combinatorial arrangement by $\mathcal{A}(\mathcal{R})$.

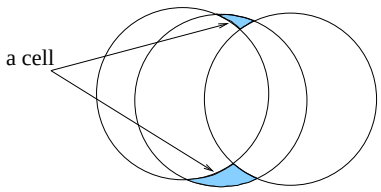


FIG. 2.1. *An arrangement of disks. The marked cell corresponds to the regions that are contained in the middle disk and only in that disk.*

The definition of a combinatorial arrangement differs from that of a topological arrangement (where one considers the subdivision into connected components induced by the ranges). For example, Figure 2.1 depicts a collection of disks. The two shadowed regions constitute a single cell in the combinatorial arrangement induced by the disk. In the definition of a topological arrangement these regions are considered as two separate cells. We often consider combinatorial arrangements of the form $(V, \mathcal{R})$, where $V \subset cells(\mathcal{R})$. We refer, in short, to combinatorial arrangements as arrangements.

2.2. Primal and dual range-spaces. Consider a range-space $(X, \mathcal{R})$. The dual set system is $(\mathcal{R}, X^*)$, where $X^* = \{N(x)\}_{x \in X} \subseteq 2^{\mathcal{R}}$ and $N(x) = \{R \in \mathcal{R} : x \in R\}$. One may represent a set system by a bipartite graph $(X \cup \mathcal{R}, E)$, with an edge (x, R) if $x \in R$. Under this representation, the dual set system corresponds to the bipartite graph in which the roles of the two sides of the vertex set are interchanged. Isomorphism of set systems is equivalent to the isomorphism between the bipartite graph representations of the corresponding set systems.

Let $\mathcal{T}$ denote a set of regions in the plane. We use $\mathcal{T}$ to denote a set of regions with some common property, for example, the set of all unit disks or the set of axis-parallel unit squares. Given a set of points X and a region R (such as a disk), when referring to R as a range (namely, a subset of X) we actually mean $R \cap X$.

A range-space $(X, \mathcal{R})$ is a $\mathcal{T}$-type range-space if $\mathcal{R} \subseteq \mathcal{T}$. We are interested in situations in which the dual of a $\mathcal{T}$-type range-space is isomorphic to a $\mathcal{T}$-type range-space.

DEFINITION 2.1. *A set of regions $\mathcal{T}$ is* self dual *if the dual range-space of every $\mathcal{T}$-type range-space is isomorphic to a $\mathcal{T}$-type range-space.*

For example, it is not hard to verify that the set of all unit disks is self dual. On the other hand, the set of all disks (or even disks of different radius) is not self dual.

The following claim states a condition on $\mathcal{T}$ that is sufficient for $\mathcal{T}$ to be self dual when X is a set of points in the plane.

CLAIM 2.2. *Let C be a fixed centrally symmetric region in the plane, and let $\mathcal{T}$ be the set of all regions congruent (via translation, not rotation) to C. Then $\mathcal{T}$ is self dual.*

Proof. Given a $\mathcal{T}$-type range-space $(X, \mathcal{R})$, let Y denote the set of centers of the ranges in $\mathcal{R}$. For a point x in the plane, let $C(x)$ denote the region congruent to C that is centered at x. For a set X of points in the plane, let $\mathcal{C}(X) \triangleq \{C(x) : x \in X\}$ denote the set of regions congruent to C centered at points of X. The range-space $(Y, \mathcal{C}(X))$ is obviously a $\mathcal{T}$-type range-space. To see that this system is isomorphic to the dual range-space $(\mathcal{R}, X^*)$, we identify every range $R \in \mathcal{R}$ with its center. Since C is centrally symmetric, it follows that $y \in C(x)$ if and only if $x \in C(y)$ for every two points x, y. This means that a center $y \in Y$ is in $C(x)$ if and only if the range $C(y)$ contains the point $x \in X$. Hence, for every point $x \in X$, the set $C(x) \cap Y$ equals the set of centers of ranges in $N(x)$, and the claim follows. □

As a corollary of Claim 2.2 we obtain the following.

COROLLARY 2.3. *Let C be a fixed centrally symmetric region in the plane, and let $\mathcal{T}$ be the set of all regions congruent (via translation, not rotation) to C. Then the CF-coloring arrangement of $\mathcal{T}$-type regions is equivalent to CF-coloring points with respect to a $\mathcal{T}$-type set of ranges.*

We rely on Corollary 2.3 in the proof of part 2 of Theorem 1.2 and in the proof of Theorem 1.5.

3. CF-coloring points with respect to ranges. In this section we present CF-coloring algorithms for points with respect to ranges. The colorings require $O(\log n)$ colors, where n denotes the number of points.

3.1. Intuition. We begin this subsection by presenting a high level description of our algorithm. We then briefly discuss how it can be applied to the special case where X is a set of n points in the plane and the ranges in $\mathcal{R}$ are intersections of X with disks.

The algorithm works in an iterative manner, where in iteration i it selects the subset of points that are colored by color number i. Let X_i denote the set of points that are colored by the color i, and let $X_{<i}$ (respectively, $X_{\leq i}$) denote the set $\bigcup_{j<i} X_j$ (respectively, $\bigcup_{j\leq i} X_j$). When determining X_i, the algorithm ensures that the following condition holds:

> For every range $S \in \mathcal{R}$, either (i) S can be served by a point colored $j < i$ (i.e., there exists $j < i : |S \cap X_j| = 1$), or (ii) $S \cap X_i$ contains at most one point, or (iii) S contains a point that is not colored yet (i.e., $S \not\subseteq X_{\leq i}$).

Correctness follows because if either (i) or (ii) holds, then S can be served by a point colored $j \leq i$, while if neither (i) nor (ii) holds, then there will be a point colored by a color greater than i that can serve S. In fact, a coloring that obeys the above condition has the following property: For every range $S \in \mathcal{R}$, the highest color of a point contained in S has multiplicity 1.

Observe that we can trivially obey the above condition by selecting X_i to consist of a single point in $X \setminus X_{<i}$, so that each point is colored by a different color. However, the total number of colors in this case is $|X| = n$, while we are interested in using only $O(\log n)$ colors. To obtain $O(\log n)$ colors, we show that (for the sets of ranges we consider), in each stage it is possible to select at least a constant fraction of the

remaining points (i.e., $|X_i| \geq \frac{1}{4} \cdot |X \setminus X_{<i}|$).

To make the above more concrete, consider the special case of coloring a set of points X with respect to disks. First assume that the points all lie on a straight line. In such a case, the choice of X_i involves simply picking every other point of $X \setminus X_{<i}$ (see Figure 3.1). By convexity, if a disk D contains two (or more) points from X_i, then it must contain all the points in between these two points. Between every two points in X_i there must exist at least one point not in $X_{\leq i}$. It follows that the condition required from X_i holds. Since $|X_i| = \lceil \frac{|X \setminus X_{<i}|}{2} \rceil$, the number of colors used is $O(\log n)$, as desired.

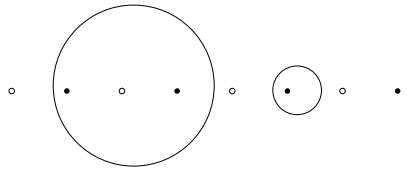


FIG. 3.1. *Selection of X_i when the points lie on a straight line. The points drawn are those in $X \setminus X_{<i}$. The points of X_i are denoted by filled dots. The unfilled dots denote points in $X \setminus X_{\leq i}$. The disk on the left contains two points in X_i and hence also an unfilled dot. The disk on the right contains only a single point in X_i.*

The choice of X_i when the points are in general position in the plane is more involved. In this case, we construct the Delaunay graph G_i of the set $X \setminus X_{<i}$: Two points p_i and p_j form an edge in G_i if and only if there is a closed disk D that contains p_i and p_j on its boundary and does not contain any other point in $X \setminus X_{<i}$. The graph G_i is planar and hence is 4-colorable. The largest color class contains at least $\frac{1}{4}$ of the remaining points and is an independent set in G_i. In Claim 3.4 below we prove that the largest color class is a good candidate for X_i.

3.2. A general framework. We start by presenting a general framework for CF-coloring a set X of points with respect to a set $\mathcal{R} \subseteq 2^X$ of ranges, and describe sufficient conditions under which the resulting coloring uses $O(\log n)$ colors. Since every range $R \in 2^X$ that contains a single point from X is trivially served by that point, we assume that every range in $\mathcal{R}$ contains at least two points from X.

DEFINITION 3.1. *Let X be a set of points and $\mathcal{R} \subseteq 2^X$ a set of ranges. A partition (X_1, X_2) of X is* $\mathcal{R}$-useful *if $X_1 \neq \emptyset$ and*

$$\forall S \in \mathcal{R} \ : \ |S \cap X_1| = 1 \quad \textit{or} \quad S \cap X_2 \neq \emptyset.$$

ALGORITHM 1. CF-COLOR$(X, \mathcal{R})$—CF-color a set X with respect to a set of ranges $\mathcal{R}$.

1: **Initialization:** $i \leftarrow 1$, $X^1 \leftarrow X$, $\mathcal{R}^1 \leftarrow \mathcal{R}$. ($i$ denotes an unused color, X^i is the set of points not yet colored, and $\mathcal{R}^i$ is the set of ranges that contain more than one point in X^i and cannot be served by points colored j, for $j < i$.)
2: **while** $X^i \neq \emptyset$**, do**
3: **Find an $\mathcal{R}^i$-useful partition** (X_1, X_2) of X^i. (See Claim 3.4 below.)
4: **Color:** $\forall x \in X_1 \ : \ \chi(x) \leftarrow i$.
5: **Project:** $X^{i+1} \leftarrow X_2$ and $\mathcal{R}^{i+1} \leftarrow \{S \cap X_2 : S \in \mathcal{R}^i, \ |S \cap X_1| \neq 1, \text{ and } |S \cap X_2| \geq 2\}$.
6: **Increment:** $i \leftarrow i + 1$.
7: **end while**

CLAIM 3.2. *The coloring of X computed by* CF-COLOR$(X, \mathcal{R})$ *is a CF-coloring of X with respect to $\mathcal{R}$.*

Proof. Consider a range $S \in \mathcal{R}$. Let i denote the last iteration in which $X^i \cap S \in \mathcal{R}^i$. In other words, in the ith iteration, the $\mathcal{R}^i$-useful partition (X_1, X_2) of X^i satisfies either $|X_1 \cap S| = 1$ or $|X_2 \cap S| = 1$. In the first case, S can be served by the single element $x \in X_1 \cap S$ (which is colored i). In the second case, S can be served by the single element $x \in X_2 \cap S$ (which is colored j, for $j > i$). Observe that if at the end of iteration i the range-space $\mathcal{R}^{i+1}$ becomes empty while X^{i+1} is not empty, then the partition $(X^{i+1}, \emptyset)$ is trivially $\mathcal{R}^{i+1}$-useful, and all the points in X^{i+1} can be colored with the color $i + 1$. □

Note that Algorithm CF-COLOR computes a CF-coloring in which every range $S \in \mathcal{R}$ is served by the point with the highest color among the points in S.

3.2.1. Sufficient conditions for using $O(\log |X|)$ colors. If in every iteration i we have $|X_1| = \Omega(|X^i|)$, then Algorithm CF-COLOR uses $O(\log |X|)$ colors. We formalize a condition guaranteeing that $|X_1|$ is a constant fraction of $|X^i|$. The condition is phrased in terms of a special graph that is attached to the range-space $(X^i, \mathcal{R}^i)$.

We refer to ranges $S \in \mathcal{R}^i$ as *minimal* if they are minimal with respect to inclusion. Recall that we initially assume that for every $S \in \mathcal{R}$, $|S| \geq 2$ (since ranges of size one are served trivially). The algorithm ensures that, in each iteration, every range in $\mathcal{R}^i$ contains at least two points. Hence, minimal ranges contain at least two points.

DEFINITION 3.3. *A* Delaunay graph *of a set system $(X, \mathcal{R})$ is a graph $DG_{\mathcal{R}}(X, E)$, defined as follows. For every minimal $S \in \mathcal{R}$, pick a pair $u, v \in S$ and define $e(S) = (u, v)$. The edge set E is defined by $E = \{e(S) : S \in \mathcal{R}$ and S is minimal$\}$.*

A Delaunay graph of a set system is not uniquely defined if there exist minimal ranges that contain more than two points. To simplify the presentation, we abuse notation and refer to the Delaunay graph of a set system as if it were unique.

We now discuss how Definition 3.3 is an extension of the standard definition of the Delaunay graph of a set of points in the plane. The Delaunay graph of a set of points X in the plane is defined as the dual graph of the Voronoi diagram of X [BKOS97]. Theorem 9.6 in [BKOS97] suggests an equivalent definition: Two points p_i and p_j form an edge in the Delaunay graph of X if and only if there is a closed disk D that contains p_i and p_j on its boundary and does not contain any other point in X. This equivalent definition implies that the edge set of a Delaunay graph corresponding to X equals the set of minimal ranges containing two points induced by disks. We leave it as an exercise to prove that every minimal range induced by a disk contains exactly two points. Hence, in the case of points in the plane and disks, $DG_{\mathcal{R}}(X, E)$ is the standard Delaunay graph of X.

The next claim shows how an $\mathcal{R}$-useful partition can be found by Algorithm CF-COLOR.

CLAIM 3.4. *If $X_1 \subseteq X$ is an independent set in $DG_{\mathcal{R}}$, then the partition $(X_1, X \setminus X_1)$ is $\mathcal{R}$-useful.*

Proof. Assume for the sake of contradiction that there exists an independent set X_1 such that $(X_1, X \setminus X_1)$ is not an $\mathcal{R}$-useful partition of X. That is, there exists a range $S \in \mathcal{R}$ such that $|S \cap X_1| \neq 1$ and $S \cap (X \setminus X_1) = \emptyset$. Note that assuming that $S \cap (X \setminus X_1) = \emptyset$ necessarily implies that $S \subseteq X_1$, and so we may replace the first condition (i.e., $|S \cap X_1| \neq 1$) by $|S \cap X_1| \geq 2$.

Let S' denote a minimal range that is a subset of S (hence $S' \subseteq X_1$). By the

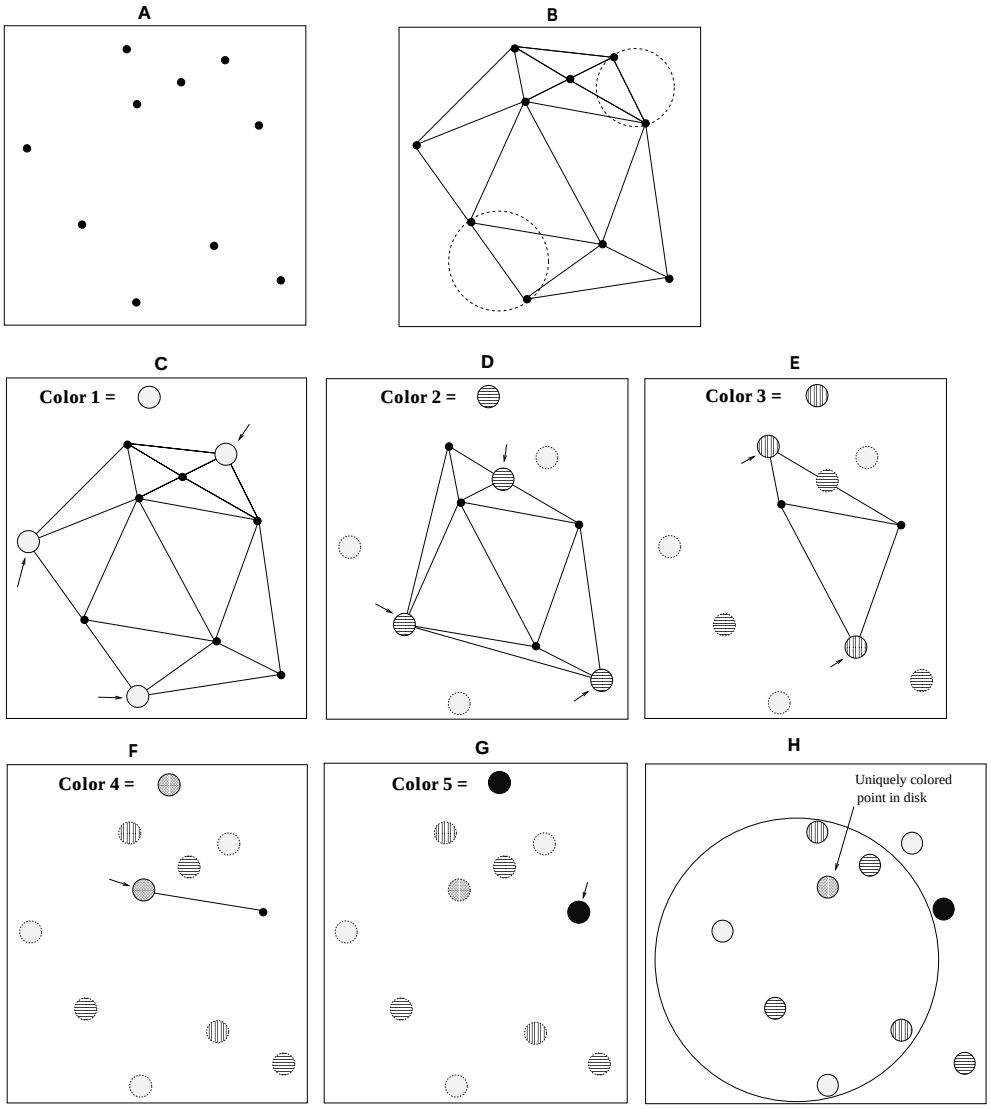


FIG. 3.2. *An illustration of the execution of Algorithm* 1 *in the case of disks in the plane. In panel A we see the given set of points. In panel B, the Delaunay graph is depicted. Recall that there is an edge between every pair of points* p, q *that are separated from the rest of the points by a disk. Two such disks are depicted for this graph. C–G depict five steps of the algorithm. In each step we see the Delaunay graph over the remaining uncolored points, where the newly colored points are marked by arrows. (The previously colored points also appear in the figure, but they are not part of the Delaunay graph.) In H we see the final coloring of all points, and an example of a disk and the point that can serve it. (In general, there may be more than one such point.)*

definition of the set of edges E in the Delaunay graph $DG_{\mathcal{R}}$ of $(X, \mathcal{R})$, it follows that there is an edge $e(S')$ between two points in S'. But this contradicts the assumption that X_1 is an independent set, and the claim follows. □

The method we use to show that Delaunay graphs have large independent sets is to show that Delaunay graphs are planar. Another easy way to show that there exists a large independent set is, for example, to show that the number of edges is linear.

CLAIM 3.5. *If in each iteration of the algorithm the Delaunay graph of* $(X^i, \mathcal{R}^i)$ *is planar, then Algorithm* 1 *uses* $O(\log |X|)$ *colors.*

Proof. By Claim 3.4, it suffices to show that, in every iteration of Algorithm CF-COLOR, the Delaunay graph has an independent set X_1 that satisfies $|X_1| = \Omega(|X^i|)$. The existence of a large independent set X_1 in the Delaunay graph $DG_{\mathcal{R}^i}(X^i, E)$ follows from the planarity of $DG_{\mathcal{R}^i}$. Planarity implies that the graph is 4-colorable [AH77a, AH77b], and therefore, the largest color-class is an independent set of size at least $|X^i|/4$. (Recall that planar graphs can be 4-colored in polynomial time [AH77a, AH77b, RSST96]. For our purposes, a coloring using six colors suffices. One could easily color planar graphs using six colors, since the minimum degree is at most 5. This means that a greedy algorithm could be used to find an independent set of size at least $|X^i|/6$.) □

In the rest of this section we apply Algorithm CF-COLOR to three types of range-spaces: disks in the plane, half-spaces in $\mathbb{R}^3$, and homothetic centrally symmetric convex regions in the plane. For each of these cases we prove that the premise of Claim 3.5 is satisfied—that is, that the Delaunay graph of the corresponding range-space is planar. Moreover, for disks, half-spaces in $\mathbb{R}^3$, and scaled translations of a convex polygon, the corresponding Delaunay graphs are computable in polynomial time, which implies that Algorithm CF-COLOR is polynomial.

3.3. Disks in the plane. Recall that, in the case of disks in the plane, the Delaunay graph that we attach to the set system $(X, \mathcal{R})$ is the standard Delaunay graph. Hence, the Delaunay graph is planar [BKOS97, Theorem 9.5]. We may now apply Claim 3.5 to obtain the following lemma.

LEMMA 3.6. *Let* X *denote a set of* n *points in the plane. Let* $\mathcal{R}$ *denote the collection of all subsets of* X *of size at least* 2 *obtained by intersecting* X *with a (closed) disk. Then it is possible to color* X *with respect to* $\mathcal{R}$ *using* $O(\log n)$ *colors.*

3.4. Half-spaces in $\mathbb{R}^3$. Given a hyperplane H (not parallel to the z-axis), the positive half-space H^+ is the set of all points that either lie on or are above H. We denote by $\mathcal{H}^+$ the set of all positive half-spaces in $\mathbb{R}^3$.

LEMMA 3.7. *Let* X *be a set of* n *points in* $\mathbb{R}^3$. *Let* $\mathcal{R}$ *denote the collection of all subsets of* X *of size at least two obtained by intersecting* X *with a half-space in* $\mathcal{H}^+$. *Then there exists a CF-coloring of* X *with respect to* $\mathcal{R}$ *that uses* $O(\log n)$ *colors.*

Let $CH(X)$ denote the convex hull of X. We make the following simplifying assumption: Every point in X is an extreme point of $CH(X)$. If not, then all the points of X that are not extreme points of $CH(X)$ may be colored by a unique "passive" color. The coloring of nonextreme points by a passive color means, in effect, that these points are removed. This reduction is justified by the fact that every half-space H^+ that intersects the convex hull of X must contain an extreme point of X. The coloring will be a CF-coloring of the extreme points of $CH(X)$ with respect to positive half-spaces, and hence $X \cap H^+$ will be served as well.

CLAIM 3.8. *Every minimal range in the range-space* $(X, \mathcal{R})$ *is a pair of points.*

Proof. Consider a range $R \in \mathcal{R}$ defined by half-space H^+. Translate H upward as much as possible so that every further translation upward reduces the range defined by the positive half-space to less than two points. Let H_1 denote the plane parallel to H obtained by this translation. Let R_1 denote the range corresponding to the positive half-space H_1^+. If R_1 contains more than two points, then either R_1 is contained in the plane H_1 or all but one of the points in R_1 are in the plane H_1. Assume that $R_1 \subset H_1$. Consider a line ℓ in H_1 that passes through two adjacent vertices u, v (i.e., an edge) in the polygon corresponding to the (two-dimensional) convex hull of R_1

relative to the plane H_1. Tilt the plane H_1 slightly, where the line ℓ serves as the axis of rotation. It is possible to rotate H_1 so that the resulting plane H_2 satisfies $X \cap H_2^+ = \{u, v\}$. A similar argument applies if there is a single point in $R_1 \setminus H_1$, and the claim follows. □

Proof of Lemma 3.7. Claim 3.8 implies that the Delaunay graph $DG_{\mathcal{R}} = (X, E)$ of the range-space $(X, \mathcal{R})$ is defined by $(u, v) \in E$ if and only if there exists a positive half-space H^+ such that $X \cap H^+ = \{u, v\}$. Recall that we assumed that every point in X is an extreme point of $CH(X)$. Two points $x, y \in X$ are *adjacent* if there exists a supporting plane H of $CH(X)$ such that $H \cap X = \{u, v\}$. The *skeleton graph* $G' = (X, E')$ of $CH(X)$ is the graph over the points in X with edges between adjacent points. The skeleton graph is drawn on the boundary of $CH(X)$ using straight lines without crossings. Since the boundary of $CH(X)$ is homeomorphic to a sphere, it follows that the skeleton graph is planar.

By definition, the edge set of the Delaunay graph is contained in the edge set of the skeleton graph. Hence the Delaunay graph is planar and, by Claim 3.5, X can be CF-colored with respect to $\mathcal{R}$ using $O(\log |X|)$ colors. □

3.5. Scaled translations of a convex region in the plane. In this subsection we prove Theorem 1.7. We first introduce some definitions and notation.

For a closed region C let ∂C denote the boundary of C, and let $\mathring{C}$ denote the interior of C. We next recall the definition of homothecy (cf. [C69, p. 68]).

DEFINITION 3.9. *A transformation $\tau : \mathbb{R}^2 \to \mathbb{R}^2$ is a* homothecy *if there exist a point O (called the* homothetic center*) and a nonzero real number λ (called the* similitude ratio*) such that*

1. *O is a fixed point of τ (namely, $O = \tau(O)$);*
2. *every point $P \neq O$ is mapped to a point $\tau(P)$ where* (i) *$\tau(P)$ is on the line OP, and* (ii) *the length of the segment $O\tau(P)$ satisfies $|O\tau(P)| = \lambda \cdot |OP|$.*

We use the notation $C' \sim C$ to denote that C' is a scaled translation of C. For a homothetic transformation $\tau : \mathbb{R}^2 \to \mathbb{R}^2$, we denote the image of a set $S \subseteq \mathbb{R}^2$ under τ by $\tau(S)$. Note that if the similitude ratio of a homothecy τ is positive, then $\tau(C) \sim C$.

DEFINITION 3.10. *A range $S \in \mathcal{R}$ is* induced *by a region C if $S = C \cap X$. A range $S \in \mathcal{R}$ is* boundary-induced *by a closed region C if $S = \partial C \cap X$ and $\mathring{C} \cap X = \emptyset$.*

Recall that, for the purpose of CF-coloring, ranges that contain one point as well as the empty range are trivial. Hence, we do not consider the empty set and subsets that contain a single point to be ranges. Therefore, we define the range-space $\mathcal{R}$ induced by a collection of regions $\mathcal{C}$ by

$$\mathcal{R} = \{C \cap X : C \in \mathcal{C} \text{ and } |C \cap X| \geq 2\}.$$

It follows that minimal ranges contain at least two points.

Let C denote a compact convex region in the plane. Let $X \subset \mathbb{R}^2$ denote a finite set of points in the plane. Let $(X, \mathcal{R})$ denote the range-space induced by the set of all scaled translations of C. By Claim 3.5, in order to prove Theorem 1.7, it suffices to prove that the Delaunay graph of $(X, \mathcal{R})$ is planar. To this end we first show the following.

CLAIM 3.11. *Every minimal range $S \in \mathcal{R}$ is boundary-induced by a region $C' \sim C$.*

Proof. Since S is a range, there exists a scaled translation $C_S \sim C$ such that $X \cap C_S = S$. By contracting C_S, if necessary, we may guarantee that the boundary of C_S contains a point from S. The interior of C_S contains at most one point of

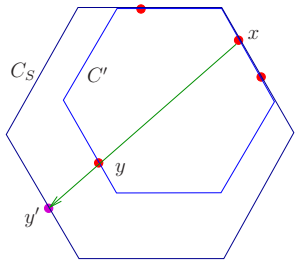


FIG. 3.3. *An illustration for the proof of Claim* 3.11.

S. Otherwise, by an infinitesimal contraction, we are left with a range $S' \subsetneqq S$ that contains at least two points, thus contradicting the minimality of S.

We now show how to find a region $C' \sim C$ such that all of S lies on the boundary of C'. Let $x \in S$ denote a point on the boundary of C_S. If S is not boundary-induced by C_S, then there is a unique point $y \in S \cap \mathring{C}_S$. The region C' is the image of C with respect to the homothecy τ that is defined as follows. Let y' denote the intersection point of the boundary of C_S with the half-open ray emanating from x toward y. Set x to be the homothetic center, and set the similitude ratio to be the ratio $|xy|/|xy'|$. By definition of τ, both x and y are on the boundary of C'. By minimality of S, it follows that $C' \cap X = S$. By definition of τ and convexity of C, it follows that $C' \subseteq C_S$. If a point $z \in S$ is in the interior of C', then it is in the interior of C_S; hence $z = y$, which contradicts $y \in \partial C'$. It follows that every point in S is in the boundary of C', and the claim follows. For an illustration, see Figure 3.3. □

We now show that a planar drawing of the Delaunay graph $DG_{\mathcal{R}} = (X, E)$ is obtained if its edges are drawn as straight line segments. Consider two edges $(x_0, y_0), (x_1, y_1) \in E$. For $i = 0, 1$, assume that $x_i, y_i \in S_i$ for a minimal range $S_i \in \mathcal{R}$, where $S_0 \neq S_1$. Let $C_i \sim C$ denote scaled translations of C such that S_i is boundary-induced by C_i. If $C_0 \cap C_1 = \emptyset$, then the segments x_0y_0 and x_1y_1 do not cross each other. If $C_0 \cap C_1 \neq \emptyset$, then the boundaries ∂C_0 and ∂C_1 intersect.

We first consider the case in which ∂C does not contain a straight side. Namely, no three points on ∂C are colinear. Under this assumption, since C_i is a scaled translation of C for $i = 0, 1$, it follows that $\partial C_0 \cap \partial C_1$ contains at most two points.

If $\partial C_0 \cap \partial C_1$ contains a single point p, then one can separate the convex regions C_0 and C_1 using a straight line passing through p. This separating line implies that the segments x_0y_0 and x_1y_1 cannot cross each other.

If $\partial C_0 \cap \partial C_1$ contains two points, denote these points by p and q. The boundary ∂C_i is partitioned into two simple curves, each delimited by the points p and q; one curve is contained in $\partial C_i \setminus \mathring{C}_{1-i}$, and the second curve is $\partial C_i \cap C_{1-i}$. We denote the curve $\partial C_i \setminus \mathring{C}_{1-i}$ by γ_i, and we denote the curve $\partial C_i \cap C_{1-i}$ by γ_i'. Since the interior $\mathring{C}_{1-i}$ lacks points of X, it follows that x_i and y_i are in γ_i.

In order to prove that the segments x_0y_0 and x_1y_1 do not cross each other, it suffices to show that the line pq separates $\gamma_0 \setminus \{p, q\}$ and $\gamma_1 \setminus \{p, q\}$. (Intersection of two edges means that the edges share an interior point, which cannot be p or q.) Assume, for the sake of contradiction, that $\gamma_0 \setminus \{p, q\}$ and $\gamma_1 \setminus \{p, q\}$ are on the same side of the line pq. These curves do not intersect, and together with the segment pq, one must contain the other, contradicting their definition.

The case in which ∂C contains a straight side (and so $\partial C_0 \cap \partial C_1$ may contain a subsegment of such a straight side) is dealt with similarly to the case in which ∂C

does not contain a straight side. It is not hard to verify that in such a case $\partial C_0 \cap \partial C_1$ consists of at most two connected components (each either straight line or a single point). By picking p to be any point from one component and q to be any point from the other component, we can apply essentially the same argument used above.

This concludes the proof of Theorem 1.7.

4. CF-colorings of arrangements of disks. In this section we prove Theorems 1.2 and 1.3 stated in the introduction.

4.1. Proof of Theorem 1.2. Part 2 of Theorem 1.2 is proved as follows. The disk centers $X \subset \mathbb{R}^2$ are given. Consider a radius r (which is not given to the algorithm!), and apply Corollary 2.3 to the arrangement $\mathcal{A}(\mathcal{S}_r(X))$. Let Y denote the set consisting of representatives from every cell in $cells(\mathcal{S}_r(X))$. The dual range-space is isomorphic to a range-space with (i) a ground set X and (ii) ranges induced by $\mathcal{S}_r(Y)$. We extend the range-space to ranges induced by all the disks (of all radiuses). A CF-coloring of the points in X with respect to the set of all disks is also a CF-coloring of every arrangement $\mathcal{A}(\mathcal{S}_r(X))$. Part 2 of Theorem 1.2 now follows directly from Lemma 3.6.

We now turn to proving part 1 of Theorem 1.2.

A transformation to points and half-spaces. In what follows, we show that the problem of CF-coloring n arbitrary disks in the plane reduces to CF-coloring of a set of points X in $\mathbb{R}^3$ with respect to the set of ranges $\mathcal{H}^+(X)$ determined by all positive half-spaces containing at least two points from X.

We use a fairly standard dual transformation that transforms a point $p = (a, b)$ in $\mathbb{R}^2$ to a plane p^* in $\mathbb{R}^3$, with the parameterization $z = -2ax - 2by + a^2 + b^2$, and transforms a disk S in $\mathbb{R}^2$, with center (x, y) and radius $r \geq 0$, to a point S^* in $\mathbb{R}^3$, with coordinates $(x, y, r^2 - x^2 - y^2)$.

It is easily seen that, in this transformation, a point $p \in \mathbb{R}^2$ lies inside (respectively, on the boundary of, outside) a disk S if and only if the point $S^* \in \mathbb{R}^3$ lies above (respectively, on, below) the plane p^*. Indeed, a point (a, b) lies inside a disk with center (x, y) and radius r if and only if $(a - x)^2 + (b - y)^2 < r^2$. After rearrangement, this is equivalent to $-2ax - 2by + a^2 + b^2 < r^2 - x^2 - y^2$. Now this inequality is equivalent to the condition that the point $(x, y, r^2 - x^2 - y^2) = S^*$ lies above the plane $z = -2ax - 2by + a^2 + b^2$, as asserted. The cases of a point lying on the boundary of a disk or outside a disk are treated analogously.

Given a collection $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n distinct disks in the plane, one can use the above transformation to obtain a collection $\mathcal{S}^* = \{S_1^*, \ldots, S_n^*\}$ of n points in $\mathbb{R}^3$ such that any CF-coloring of $\mathcal{S}^*$ with respect to $\mathcal{H}^+(\mathcal{S}^*)$, with k colors, induces a CF-coloring of the disks of $\mathcal{S}$ with the same set of k colors.

As shown in subsection 3.4 (Lemma 3.7), it is possible to apply Algorithm 1 to obtain a CF-coloring of the points in $\mathcal{S}^*$ with respect to $\mathcal{H}^+(\mathcal{S}^*)$ using $O(\log n)$ colors. Recall that part 1 of Theorem 1.2 states that the number of colors is of the order of the minimum between $\log n$ and $\sum_{i=1}^{\log(\rho)+1} \log \phi_{2^i}(\mathcal{S}^i)$. Recall that ρ is the size-ratio of $\mathcal{S}$, $\mathcal{S}^i$ is the subset of disks in $\mathcal{S}$ whose radius is in the range $[2^{i-1}, 2^i)$, and $\phi_{2^i}(\mathcal{S}^i)$ is the maximum number of disks in $\mathcal{S}^i$ whose centers reside in a common $2^i \times 2^i$ square. To obtain the latter bound we proceed in two steps: First we assume that the size-ratio is at most 2, and then we deal with the more general case.

The tiling. Assume that the size-ratio ρ is at most 2. By scaling, we may assume that every radius is in the interval $[1, 2]$. We partition the plane into 2×2 square tiles. We say that a disk S *belongs* to tile T if the center of S is in T. We denote the

subset of disks in $\mathcal{S}$ that belong to T by $\mathcal{S}(T)$. Note that the union of the disks in any given tile intersects at most nine different tiles. We assign a *palette* (i.e., a subset of colors) to each tile using nine different palettes, where the disks belonging to a particular tile are assigned colors from the tile's palette. Palettes are assigned to tiles by following a periodic 3×3 assignment. This assignment has the property that any two disks that belong to different tiles either do not intersect or their tiles are given different palettes (so that necessarily the two disks are assigned different colors). By the definition of local density we have that $|\mathcal{S}(T)| \leq \phi_2(\mathcal{S})$ for every tile T. Since we can color the set of disks $\mathcal{S}(T)$ belonging to tile T using $O(\log |\mathcal{S}(T)|)$ colors, and the total number of palettes is nine, we get the desired upper bound of $O(\log \phi_2(\mathcal{S}))$ colors. The general case of arbitrary size-ratio is dealt with by first partitioning the set of disks into classes according to their radius. The ith class, denoted $\mathcal{S}^i$, consists of disks, the radiuses of which are in the interval $[2^i, 2^{i+1})$. Within each class, the size-ratio is bounded by 2; hence we can CF-color each class using $O(\log \phi_{2^i}(\mathcal{S}^i))$ colors. By using a different (super-)palette per class, we obtain the desired bound on the number of colors, i.e., $\sum_{i=1}^{\log \rho+1} O(\log \phi_{2^i}(\mathcal{S}^i))$.

4.2. Bicriteria CF-coloring algorithms. In this section we prove Theorem 1.3. The first part of the theorem reveals a trade-off between the number of colors used and the fraction of the area that is served. The second part of the theorem reveals a trade-off between the number of colors used to serve the union of the unit disks and the radiuses of the serving disks.

We first derive the following corollary from Theorem 1.2.

COROLLARY 4.1. *Let $\mathcal{S}$ be a set of unit disks, and let $d_{\min}(\mathcal{S})$ be the minimum distance between the centers of disks in $\mathcal{S}$. If $d_{\min}(\mathcal{S}) \leq 2$, then every arrangement $\mathcal{A}(\mathcal{S})$ of unit disks can be CF-colored using $O\left(\log\left(\min\left\{|\mathcal{S}|, \frac{1}{d_{\min}(\mathcal{S})}\right\}\right)\right)$ colors.*

Observe that if $d_{\min}(\mathcal{S}) > 2$, then a single color suffices since the disks are disjoint.

Proof. Obviously $\phi_1(\mathcal{S}) \leq |\mathcal{S}|$. Since a unit square can be packed with at most $O(\frac{1}{d_{\min}(\mathcal{S}(T))^2})$ many disks of radius $d_{\min}(\mathcal{S}(T))$, it follows that $\phi_1(\mathcal{S}) = O(\frac{1}{d_{\min}(\mathcal{S}(T))^2})$. □

Let $X \subset \mathbb{R}^2$ denote a finite set of centers of disks. Recall that $\mathcal{S}_r(X) = \{B(x,r) \mid x \in X\}$, where $B(x,r)$ denotes a disk of radius r centered at x. Let $A_r(X) = \bigcup_{x \in X} B(x,r)$. The area of a region A in the plane is denoted by $|A|$. Let $L_r(X)$ denote the length of the boundary of $A_r(X)$. In order to prove Theorem 1.3 we shall need the following two lemmas, which are proved subsequently.

LEMMA 4.2. *For every finite set X of points in the plane,*

$$|A_1(X)| \geq \frac{1}{2} \cdot L_1(X).$$

LEMMA 4.3. *For every finite set X of points in the plane and every $\varepsilon > 0$,*

$$|A_{1+\varepsilon}(X) - A_1(X)| \leq (2\varepsilon + \varepsilon^2) \cdot L_1(X).$$

Proof of Theorem 1.3. We start with the second part. Let $X' \subseteq X$ denote a maximal subset with respect to inclusion such that $||x_1 - x_2|| \geq \varepsilon$ for every $x_1, x_2 \in X'$. Observe that $\bigcup \mathcal{S}_1(X) \subseteq \bigcup \mathcal{S}_{1+\varepsilon}(X')$. Corollary 4.1 implies that $\mathcal{S}_{1+\varepsilon}(X')$ can be CF-colored using $O(\log \frac{1+\varepsilon}{\varepsilon})$ colors. The second part follows.

We now turn to the first part. Let $\varepsilon_1 = \varepsilon/6$ and X' as above. Corollary 4.1 implies that there exists a CF-coloring χ of $\mathcal{S}_1(X')$ using $O\left(\log \frac{1}{\varepsilon}\right)$ colors. To complete the

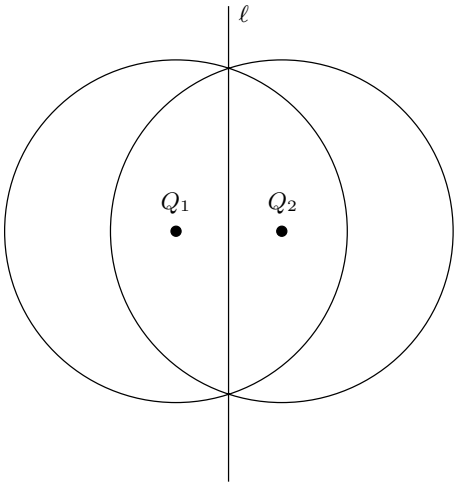


FIG. 4.1. *An illustration for the proof of Lemma* 4.4.

proof we need to show that

$$\frac{|A_1(X) - A_1(X')|}{|A_1(X)|} \leq \varepsilon.$$

Since $A_1(X) \subseteq A_{1+\varepsilon_1}(X')$ and $A_1(X') \subseteq A_1(X)$, it suffices to prove that

$$\frac{|A_{1+\varepsilon_1}(X') - A_1(X')|}{|A_1(X')|} \leq \varepsilon.$$

By Lemmas 4.2 and 4.3 it follows that

$$\frac{|A_{1+\varepsilon_1}(X') - A_1(X')|}{|A_1(X')|} \leq \frac{(2\varepsilon_1 + \varepsilon_1^2) \cdot L_1(X')}{\frac{1}{2} \cdot L_1(X')} = 4 \cdot \varepsilon_1 + 2\varepsilon_1^2.$$

Since $\varepsilon < 1$, it follows that $4 \cdot \varepsilon_1 + 2\varepsilon_1^2 \leq 6 \cdot \varepsilon_1 = \varepsilon$, and the corollary follows. □

4.2.1. Proving Lemmas 4.2 and 4.3. We denote a sector by $sect(Q, \alpha, r)$, where Q is its center, α is its angle, and r is its radius. A boundary sector of $A_1(X)$ is a sector $sect(Q, \alpha, 1)$ such that $Q \in X$ and its arc is on the boundary of $A_1(X)$. A boundary sector is maximal if it is not contained in another boundary sector. We measure angles in radians. Therefore, in a unit disk, (1) the angle of a sector equals the length of its arc, and (2) the area of a sector equals half its angle.

LEMMA 4.4. *The intersection of every two different maximal boundary sectors in* $A_1(X)$ *has zero area.*

Proof. The lemma is obvious if the boundary sectors belong to the same disk. Let $Q_1, Q_2 \in X$, and let D_i denote the circles centered at Q_i, for $i = 1, 2$, as depicted in Figure 4.1. Let $sect_i$ denote a boundary sector that belongs to circle D_i, for $i = 1, 2$. Let ℓ denote the line defined by the intersection points of the circles D_1 and D_2. The line ℓ separates the centers Q_1 and Q_2 so that they belong to different half-planes. The sector $sect_i$ is contained in the half-plane that contains Q_i, and hence $sect_1 \cap sect_2$ contains at most two points. The lemma follows. □

Proof of Lemma 4.2. The sum of the angles of the maximal boundary sectors of $A_1(X)$ equals $L_1(X)$. By Lemma 4.4, the maximal boundary sectors are disjoint, and

hence the sum of their areas is bounded by $|A_1(X)|$. However, the area of a sector of radius 1 whose angle equals α is $\alpha/2$. □

LEMMA 4.5. *Let X denote a finite set of points in the plane. For every $P \in A_{1+\varepsilon}(X) - A_1(X)$, there exists a point $Q \in X$ such that* (1) $P \in B(Q, 1+\varepsilon)$ *and* (2) *the segment $\overline{PQ}$ contains a boundary point of $A_1(X)$.*

Proof. Let Q denote a closest point in X to P. Since $P \in A_{1+\varepsilon}(X) - A_1(X)$, it follows that $P \in B(Q, 1+\varepsilon)$. Let Y denote the point at distance 1 from Q along the segment $\overline{QP}$. All we need to show is that Y is on the boundary of $A_1(X)$. If not, then Y is in the interior of a disk $B(Q', 1)$ for $Q' \in X - \{Q\}$. The triangle inequality implies that Q' is closer to P than Q, a contradiction. The lemma follows. □

Proof of Lemma 4.3. Lemma 4.5 implies that, for every point $P \in A_{1+\varepsilon}(X) - A_1(X)$, there exists boundary sector $sect(Q, \alpha, 1)$ of $A_1(X)$ (where $Q \in X$) such that

$$P \in sect(Q, \alpha, 1+\varepsilon) - sect(Q, \alpha, 1).$$

It follows that

$$\begin{aligned} |A_{1+\varepsilon}(X) - A_1(X)| &\leq \sum_{sect(Q,\alpha,1)} |sect(Q, \alpha, 1+\varepsilon) - sect(Q, \alpha, 1)| \\ &= \sum_{sect(Q,\alpha,1)} \alpha \cdot (2\varepsilon + \varepsilon^2), \end{aligned}$$

where $sect(Q, \alpha, 1)$ ranges over all maximal boundary sectors of $A_1(X)$. The claim follows by observing that the sum of the angles of the boundary sectors of $A_1(X)$ equals $L_1(X)$. □

5. Proof of Theorem 1.5. Theorem 1.5 follows from Theorem 1.7 similarly to the way that part 2 of Theorem 1.2 was shown to follow from Lemma 3.6.

Specifically, let C be a centrally symmetric convex region with a center point O, and X the set of centers that we are given. Consider a particular scaling factor r, and apply Corollary 2.3 to the arrangement $\mathcal{A}(\mathcal{C}_{r,O}(X))$. Let Y denote the set consisting of representatives from every cell in $cells(\mathcal{C}_{r,O}(X))$. The dual range-space is isomorphic to a range-space with (i) a ground set X and (ii) ranges induced by $\mathcal{C}_{r,O}(Y)$. We extend the range-space to ranges induced by all scaled translations of C. A CF-coloring of the points in X with respect to all scaled translations of C is also a CF-coloring of every arrangement $\mathcal{A}(\mathcal{C}_{r,O}(X))$. Theorem 1.5 now follows directly from Theorem 1.7.

6. Chains and CF-coloring of chains. In this section we introduce a combinatorial structure that we call a *chain*. Chains are used to establish the tightness of Theorem 1.2. They are also central to our $O(1)$ approximation algorithms for rectangles and hexagons.

6.1. Combinatorial structure. Consider an arrangement $\mathcal{A}(\mathcal{S})$ of a collection of regions in the plane $\mathcal{S}$. We associate with every cell $v \in cells(\mathcal{S})$ the subset $N(v) \subseteq \mathcal{S}$ of regions that contain the cell, namely, $N(v) = \{S \in \mathcal{S} : v \subseteq S\}$.

A set $\mathcal{S}$ of regions in the plane is said to be *indexed* if the regions are given indexes from 1 to $|\mathcal{S}|$. In the following definition we identify a region with its index. We refer to a set $\{i, i+1, \ldots, j\}$ of consecutive integers as an *interval* and denote it by $[i, j]$.

DEFINITION 6.1. *Let $\mathcal{S}$ denote an indexed set of n regions. The arrangement $\mathcal{A}(\mathcal{S})$ satisfies the* interval property *if $N(v)$ is an interval $[i, j] \subseteq [1, n]$ for every cell $v \in cells(\mathcal{S})$.*

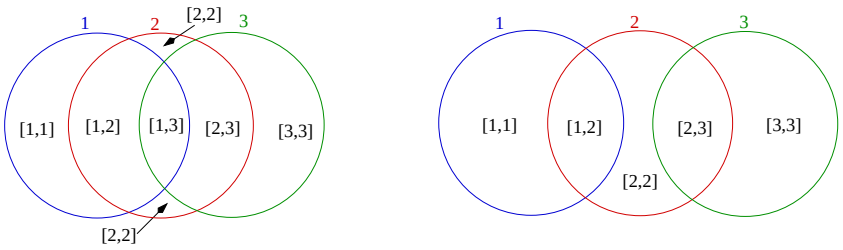


FIG. 6.1. *On the left is an arrangement of three disks that satisfies the full interval property, and on the right is an arrangement that satisfies the interval property but not the full interval property. In particular, in both cases the disks are unit disks and their centers reside on a line. However, in the arrangement on the right, the first and the third disk do not intersect, and hence there is no cell* v *such that* $N(v) = [1, 3]$.

The arrangement $\mathcal{A}(\mathcal{S})$ *satisfies the* full interval property *if it satisfies the interval property and if, in addition, for every interval* $[i, j] \subseteq [1, n]$*, there exists a cell* $v \in$ *cells*$(\mathcal{S})$ *such that* $N(v) = [i, j]$.

For an illustration of the interval and full interval properties, see Figure 6.1. The definition of the (full) interval property is sensitive to the indexing. Indexes of regions are usually based on the order of appearance of the regions along the boundary of the union of the regions. We refer, in short, to an arrangement of an indexed set of regions that satisfies the full interval property as a *chain*.

The definition of a chain implies that an arrangement $\mathcal{A}(\mathcal{S})$ is a chain if and only if the dual range-space is isomorphic to $(\{1, \ldots, n\}, \{[i, j] : 1 \leq i \leq j \leq n\})$, where $n = |\mathcal{S}|$. The next lemma, which follows directly from this observation, shows that the chain property is hereditary.

LEMMA 6.2. *Let* $\mathcal{S}$ *denote an indexed set of regions. Let* $\mathcal{S}' \subseteq \mathcal{S}$*, and let the indexes of regions* $\mathcal{S}'$ *agree with their order in* $\mathcal{S}$*. If* $\mathcal{A}(\mathcal{S})$ *is a chain, then* $\mathcal{A}(\mathcal{S}')$ *is also a chain.*

Before discussing colorings of chains, we observe that it is easy to construct chains. Consider a set $\mathcal{S}$ of n unit disks with centers positioned along a straight line at distance $\frac{1}{n+1}$ apart. Index the disks from 1 to n according to the position of their centers from left to right. The arrangement $\mathcal{A}(\mathcal{S})$ is depicted on the top of Figure 6.2. Observe that every two disks in the arrangement intersect.

We apply duality to prove that the arrangement $\mathcal{A}(\mathcal{S})$ is a chain. The arrangement is the range-space $(\mathit{cells}(\mathcal{S}), \mathcal{S})$. Let X denote a set of representatives of cells in $\mathit{cells}(\mathcal{S})$, and let Y denote the centers of unit disks in $\mathcal{S}$. The dual range-space is the pair $(Y, \{N(x)\}_{x \in X})$. Since the disks are unit disks, it follows that $N(x)$ is the intersection of Y with a unit disk centered at x. The set Y is indexed, and its points are located along a line sufficiently close so that they are included in a unit disk. Hence the collection of sets $\{N(x)\}_{x \in X}$ is simply the set of all intervals $[i, j] \subseteq [1, n]$. It follows that the arrangement $\mathcal{A}(\mathcal{S})$ is a chain, as claimed. For an illustration, see Figure 6.2 (bottom).

6.2. CF-colorings of chains. In this subsection we show that the number of colors both necessary and sufficient for CF-coloring a chain of n regions is $\Theta(\log n)$.

LEMMA 6.3. *Every CF-coloring of a chain of* n *regions uses* $\Omega(\log n)$ *colors.*

Proof. Let $I_{a,b}$ denote the set $\{[i, j] : a \leq i \leq j \leq b\}$, namely, the set of all subintervals of $[a, b]$. By definition, the dual range-space of a chain is isomorphic to the range-space $([1, n], I_{1,n})$. Therefore, CF-coloring a chain is equivalent to CF-

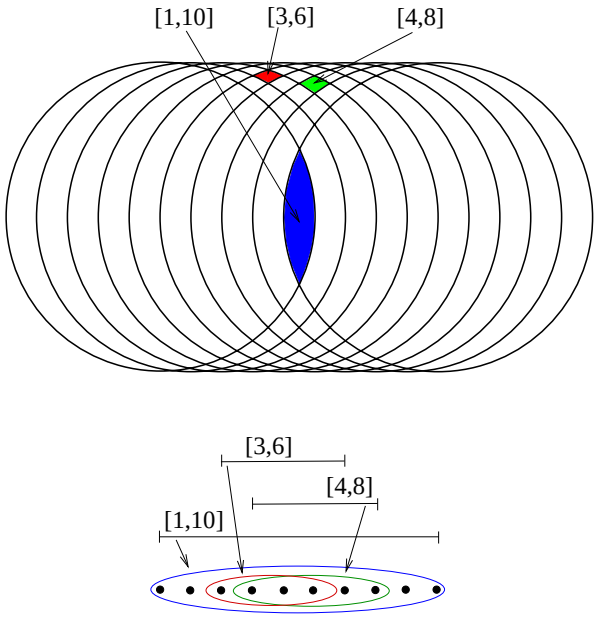


FIG. 6.2. *On the top is a chain of disks, where the disks are numbered* $1, \ldots, 10$ *from left to right. The three cells that are marked correspond to the three respective intervals. On the bottom is an illustration of the dual range-space. In the dual space there is a point for every disk in the primal space, and a subset for every cell in the primal space. Since the cells in the primal space correspond to intervals, the subsets in the dual space correspond to intervals* $[i, j] = \{i, i+1, \ldots, j\}$ *as well.*

coloring $[1, n]$ with respect to $I_{1,n}$. We hence focus on the latter problem. Let $f(n)$ denote the minimum number of colors required for such a coloring.

Consider an optimal CF-coloring χ_n of $[1, n]$ with respect to $I_{1,n}$. Let i denote the index that serves the interval $[1, n]$. It follows that for every index $j \neq i$, $\chi(j) \neq \chi(i)$. Since $\chi(i)$ is unique, it follows that every subinterval that contains i can be served by i.

We partition $I_{1,n}$ into three sets as follows: (i) $I_{1,(i-1)}$, the set of all subintervals of $[1, i-1]$; (ii) I', the set of all subintervals of $[1, n]$ that contain i; and (iii) $I_{(i+1),n}$, the set of all subintervals of $[i+1, n]$. (Observe that if $i = 1$ (respectively, $i = n$), then $I_{1,(i-1)}$ (respectively, $I_{(i+1),n}$) is empty.)

Since i can serve only intervals in I', we are left with two range-spaces that are the dual of (shorter) chains. Namely, the range-space $([1, (i-i)], I_{1,(i-1)})$ and the range-space $([(i+i), n], I_{(i+1),n})$.

Since $\chi(j)$ must differ from $\chi(i)$ for every $j \neq i$, it follows that $f(n)$ satisfies the following recurrence equation:

$$f(n) \geq 1 + \max_i \{f(i-1), f(n-i)\}.$$

Therefore, $f(n) = \Omega(\log n)$, and the lemma follows. □

LEMMA 6.4. *Every indexed arrangement of* n *regions that satisfies the interval property can be CF-colored with* $O(\log n)$ *colors.*

It suffices to prove the above lemma for chains. (In terms of the dual range-space, this simply means that we add constraints.) In fact, in section 3.1 we already presented a proof of the above lemma in the special case of unit disks whose centers

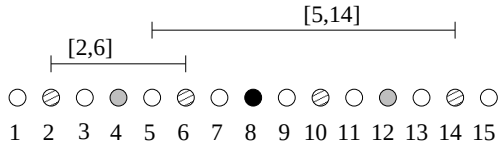


FIG. 6.3. *An illustration for Lemma* 6.4. *The* 15 *points in the figure are the points of the dual range-space. Point number* 8 *is colored with the "highest color" (corresponding to the densest filling in the illustration). Points* 4 *and* 12 *are colored with the next highest color, and points* 2, 6, 10, *and* 14 *with the next. The remaining points (all odd-numbered points) are colored with the lowest color. If we now consider, for example, the intervals (ranges in the dual space)* $[5, 14]$ *and* $[2, 6]$, *then the first can be served by point number* 8, *and the latter by point number* 4.

reside on a line. In section 6.1 we showed that a chain can be obtained from an arrangement of unit disks whose centers are collinear. Hence, the lemma follows. We provide an alternative proof of the above lemma that follows the spirit of the proof of the lower bound stated in Lemma 6.3.

Proof. We use the same notation as in the proof of the previous lemma. Without loss of generality the dual range-space is isomorphic to $([1, n], I_{1,n})$. (Adding ranges does not make CF-coloring a set of points with respect to a set of ranges any easier.) Hence, we focus on CF-coloring of such a dual range-space.

We show by induction that $f(n) \leq \lceil \log n \rceil + 1$ (see Figure 6.3). The induction basis $n = 1$ is trivial. For $n > 1$, let $i = \lceil n/2 \rceil$ and color it with the color $\lceil \log n \rceil$. The index i serves all the subintervals of $[1, n]$ that contain i. A subinterval of $[1, n]$ that does not contain i is either in $I_{1,(i-1)}$ or in $I_{(i+1),n}$. The induction hypothesis implies that the range-spaces $([1, (i-1)], I_{1,(i-1)})$ and $([(i+1), n], I_{(i+1),n})$ can each be colored by $1 + \lceil \log(n/2) \rceil = \lceil \log n \rceil$ colors. Since the ground sets of these range-spaces are disjoint, we may use the same set of colors for each. It follows that at most $\lceil \log n \rceil + 1$ colors are used, as required. □

7. An approximation algorithm for rectangles. In this section we prove Theorem 1.4 for the case of axis-parallel rectangles. For simplicity, most of the proof deals with the special case of axis-parallel unit squares. In section 7.4 we point out the modifications required for rectangles.

We begin with a high level description of the algorithm (for the special case of axis-parallel unit squares). The algorithm starts by partitioning the plane into square tiles of side-lengths 1/2. Given a set of $\mathcal{S}$ of unit squares, we say that a square $s \in \mathcal{S}$ belongs to a tile if its center resides inside the tile. Hence the tiling induces a partition of $\mathcal{S}$. We first observe that squares that belong to sufficiently distant tiles do not intersect. Therefore, as shown for the case of disks, we may assign each tile a palette of colors so that the total number of palettes used is constant, and any two different tiles whose squares may intersect are assigned different palettes. At this point we could simply apply Theorem 1.5 to separately color the squares that belong to each tile. This would give us a CF-coloring that uses $O(\log \phi(\mathcal{S}))$ colors, where $\phi(\mathcal{S})$ is the maximum number of centers of squares in $\mathcal{S}$ that are contained in a square tile of side-lengths 1/2. However, the resulting coloring may be far from optimal (recall that we are interested in a constant-ratio approximation algorithm). The reason is that squares whose centers reside in different, but neighboring, tiles may interact with each other in a manner that allows us to save in the number of colors used. For an illustration, see Figure 7.1.

Instead of coloring all squares as suggested above, our algorithm selects only a

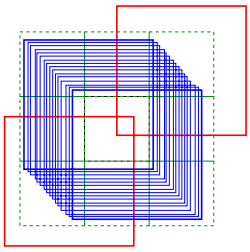


FIG. 7.1. *An example illustrating how, by taking into account intersections between squares that belong to different tiles, we may significantly reduce the number of colors required in a CF-coloring. Here there is a large number of squares that belong to the middle tile and constitute a chain. If we color the squares of each tile separately, the number of colors used is logarithmic in the size of the chain. However, there is a CF-coloring that uses only five colors: Simply color each of the thick squares by a distinct color and use the fifth color for the remaining squares.*

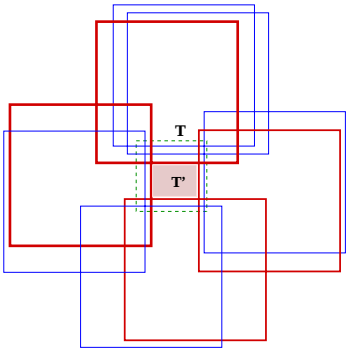


FIG. 7.2. *An illustration of the selection of squares that intersect an orphan tile T with an edge. The tile T is the dashed square, the selected squares are marked in bold, and the remaining unserved region T' is shaded.*

subset of squares, which are "essential" for serving the area covered by the union of the squares. Once this stage is over, we can "return" to each tile from which squares were requested, and color the requested squares by applying Theorem 1.5. The notion of essentialness is formalized later on in this section. It enables us to show that the total number of colors used is indeed necessary, up to a constant. Clearly, every tile that contains the center of at least one square can be completely served by any one of the squares that belong to it. Thus the main issue is serving tiles that lack centers of squares. We refer to such tiles as "orphan" tiles. In what follows we describe how an orphan tile selects the squares that are used to serve it.

Consider an orphan tile T (for which the set of squares that intersect it is nonempty). The squares that intersect it (and may hence serve parts of it) can be partitioned into two types: those that intersect it with an edge and those that intersect it with a corner. Each type can be further partitioned into four subtypes according to the edge type (respectively, corner type) with which they intersect T. We first observe that, within each subtype of squares that intersect T with an edge, we can select a single square that can serve the entire area within T that is covered by squares of this subtype. After selecting one square from each subtype, we are essentially left with the problem of serving a rectangular region, denoted by T', that is contained in T. For an illustration, see Figure 7.2.

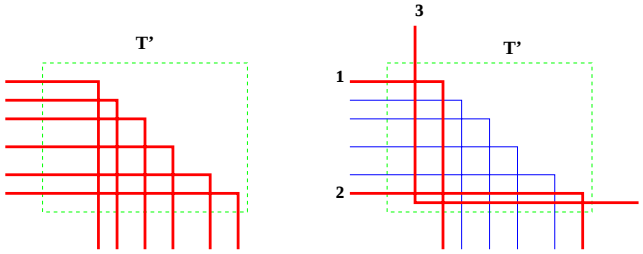


FIG. 7.3. *On the left is a subset of squares that intersect the rectangular region T' with their top-right corner. In order to cover the intersection of T' with their union it is necessary to select all squares. On the right is the same set of squares, together with one more square, which intersects T' with its bottom-left corner. Now only the 3 bold squares (labeled 1, 2, and 3) are necessary.*

Suppose that we now separately consider each subset of squares that intersect T' with a common corner type (e.g., top-right). For each corner type, it suffices to focus on the subset of squares that participate in the envelope of the squares that intersect T'. However, the envelopes of squares corresponding to different corner types may intersect in T'. Such intersections may help in reducing the number of squares needed to cover the intersection of T' with the union of all squares. For an illustration, see Figure 7.3.

In order to address the issue of "intersections" between envelopes of subsets of squares corresponding to different corner types, we consider these subsets of squares in pairs. Specifically, we first deal with "adjacent" pairs whose corresponding corners have a common edge (e.g., top-right and top-left), and then with "opposite" pairs (top-right with bottom-left, and bottom-right with top-left). For the first class of adjacent pairs, we show that by selecting at most two squares per pair we can serve the region of the intersection of each pair. For the second class of "opposite" pairs, we describe a procedure that selects a subset of squares that is at most a constant factor larger than necessary. Further details for these more involved steps are given in subsections 7.2 and 7.3.

7.1. Preliminaries. Let $\mathcal{R}$ be a set of axis-parallel rectangles of side-length at least 1. We denote a set of axis-parallel unit squares by $\mathcal{S}$. For simplicity, we assume that the rectangles (respectively, squares) in $\mathcal{R}$ (respectively, $\mathcal{S}$) are arranged in general position (i.e., no two corners of two distinct rectangles have the same x-coordinate or y-coordinate). Let $\Gamma = \{\urcorner, \ulcorner, \lrcorner, \llcorner\}$ denote the set of corner *types*. We denote the top-right corner of a rectangle R by $\urcorner(R)$. In general, for a corner $\gamma \in \Gamma$, we denote the γ-corner of R by $\gamma(R)$. The x-coordinate (y-coordinate) of a γ-corner of a rectangle R is denoted by $x_\gamma(R)$ ($y_\gamma(R)$). Let $\mathrm{op} : \Gamma \to \Gamma$ denote the permutation that swaps opposite corners (i.e., $\mathrm{op} = (\llcorner, \urcorner)(\ulcorner, \lrcorner)$). The *center* of a rectangle R is the intersection point of its two main diagonals.

The tiling. We partition the plane into "half-open" square tiles having side-lengths $1/2$, namely, $T_{i,j} = [i/2, (i+1)/2) \times [j/2, (j+1)/2)$. We say that a rectangle R *belongs* to tile T if the center of R is in T. We denote the set of rectangles in $\mathcal{R}$ that belong to tile T by $\mathcal{R}(T)$. A tile T is an *orphan* if $\mathcal{R}(T) = \emptyset$. A tile is *bare* if no rectangle in $\mathcal{R}$ intersects it. We say that two tiles are *e-neighbors* (respectively, *v-neighbors*) if they share an edge (respectively, a corner). The v-neighbor of T that shares its γ-corner with the $\mathrm{op}(\gamma)$ corner of T is denoted T_γ.

Tiles are half-open, and their side-length is defined to be half the minimum side-

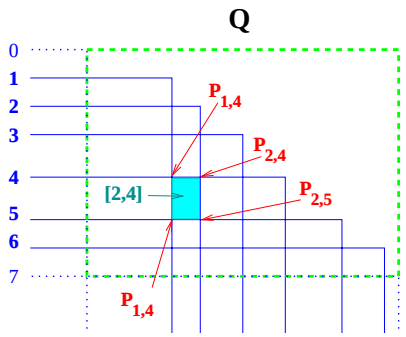


FIG. 7.4. *An illustration of a corner-chain consisting of six rectangles (the two dotted rectangles, numbered* 0 *and* 7*, are only for the sake of the analysis in the proof of Claim* 7.3*). The cell corresponding to the interval* $[2,4]$ *is filled, and its four defining corners are marked.*

length of a rectangle so that (i) if a rectangle R belongs to a tile T, then rectangle R covers the tile T; and (ii) a tile can contain at most one corner of a rectangle.

In the case of a set $\mathcal{S}$ of unit squares, squares belonging to T_γ intersect T with their γ-corner. Moreover, the corners of a unit square $S \in \mathcal{S}(T)$ reside in v-neighbors of T. Hence, a square S intersects only the tile it belongs to and the neighbors of that tile.

Corner-chains. We next consider chains determined by rectangles having the same corner in a common region. Let T be a fixed tile, and let $Q \subseteq T$ denote a rectangle. Let $\gamma \in \Gamma$ denote a corner type. Let $\mathcal{R}(Q,\gamma)$ denote the set of rectangles $R \in \mathcal{R}$ that satisfy $\gamma(R) \in Q$. The size of the tile T implies that every rectangle of side length at least 1 has at most one corner in T. Define the *Q-envelope* of $\mathcal{R}(Q,\gamma)$ to be the boundary of $\mathcal{R}(Q,\gamma)$ that is in Q (see Figure 7.4). The vertices of a Q-envelope are either corners $\gamma(R)$, for $R \in \mathcal{R}(Q,\gamma)$, or intersections of sides of two rectangles. Let $\widetilde{\mathcal{R}}(Q,\gamma)$ denote the subset of rectangles in $\mathcal{R}(Q,\gamma)$ that participate in the Q-envelope of $\mathcal{R}(Q,\gamma)$.

The next claim shows that the corner γ determines whether the Q-envelope is nonincreasing or nondecreasing.

CLAIM 7.1. *Let Q be a rectangular region with side-lengths at most* $1/2$*. If $\gamma \in \{\llcorner, \urcorner\}$, then the Q-envelope of $\mathcal{R}(Q,\gamma)$ is nonincreasing, and if $\gamma \in \{\ulcorner, \lrcorner\}$, then the Q-envelope of $\mathcal{R}(Q,\gamma)$ is nondecreasing.*

Proof. We prove the claim for $\gamma = \urcorner$. An analogous argument holds for the other cases. Let $R_1, \ldots, R_m$ $(m = |\widetilde{\mathcal{R}}(Q,\gamma)|)$ be an ordering of $\widetilde{\mathcal{R}}(Q,\gamma)$ which satisfies $x_\urcorner(R_1) < x_\urcorner(R_2) < \cdots < x_\urcorner(R_m)$. We show that $y_\urcorner(R_1) > y_\urcorner(R_2) > \cdots > y_\urcorner(R_m)$. Assume, in contradiction, that for some pair of squares $R_k, R_\ell \in \widetilde{\mathcal{R}}(Q,\gamma)$, where $k < \ell$ (so that $x_\urcorner(R_k) < x_\urcorner(R_\ell)$), we have that $y_\urcorner(R_k) < y_\urcorner(R_\ell)$. In such a case we would have that $(x_\urcorner(R_k), y_\urcorner(R_k)) \in R_\ell$, contradicting the fact that R_k belongs to the envelope $\widetilde{\mathcal{R}}(Q,\gamma)$. □

The next definition will be useful in all that follows.

DEFINITION 7.2. *Let Q be a region, and let $\mathcal{S}$ be an indexed set of regions. Let $\mathcal{S}_Q$ denote the set of regions $\{Q \cap S\}_{S \in \mathcal{S}}$. Assume that each region $Q \cap S \in \mathcal{S}_Q$ inherits the index of S. We say that* $\mathcal{S}$ is a chain with respect to Q *if the arrangement $\mathcal{A}(S_Q)$ is a chain.*

CLAIM 7.3. *Let Q be a rectangular region with side-lengths at most* $1/2$*. Index the*

rectangles of $\widetilde{\mathcal{R}}(Q,\gamma)$ according to the x-coordinate of their γ-corner. Then $\widetilde{\mathcal{R}}(Q,\gamma)$ is a chain with respect to Q.

Claim 7.3 justifies referring to $\widetilde{\mathcal{R}}(Q,\gamma)$ as a *corner-chain*.

Proof. We prove the claim for $\gamma = \urcorner$. The other three cases can be reduced to this case by "turning the picture." Let $R_1,\ldots,R_m$ ($m = |\widetilde{\mathcal{R}}(Q,\gamma)|$) be an ordering of $\widetilde{\mathcal{R}}(Q,\gamma)$ according to the x coordinates of their $\urcorner$-corner. Let R_0 and R_{m+1} be two "fictitious" rectangles, where the right side of R_0 coincides with the left side of Q, and the top side of R_{m+1} coincides with the bottom side of Q. Let $P_{i,j}$, for $0 \le i \le j \le m+1$, denote the intersection of the right side of R_i and the top side of R_j. Note that for every $1 \le i \le m$, $P_{i,i} = \urcorner(R_i)$, $P_{i,m+1}$ is the intersection of R_i with the bottom side of Q, and $P_{0,i}$ is the intersection of R_i with the left side of Q. By Claim 7.1, it follows that $P_{i,j}$ is well defined and that $P_{i,j} \in Q$ for every $1 \le i \le j \le m+1$. The arrangement of $\widetilde{\mathcal{R}}(Q,\gamma)$ in Q is a set of rectangular-shaped cells, the corners of which are the set of points $\{P_{i,j}\}$. Specifically, for every $1 \le i \le j \le m$, the cell v for which $N(v) = [i,j]$ is the rectangle whose corners are $P_{i-1,j+1}$, $P_{i-1,j}$, $P_{i,j}$, and $P_{i,j+1}$. ∎

Disjoint palettes. In the case of unit squares we assign a palette (i.e., a subset of colors) to each tile, using in total nine disjoint palettes. Palette distribution is such that neighboring tiles are assigned different palettes (i.e., we periodically assign nine different palettes to blocks of 3×3 tiles). The tile size implies that if two squares belong to different tiles that are assigned the same palette, then the squares have an empty intersection.

7.2. Main lemmas. In this section we lay the ground for our algorithm and its analysis by presenting our main lemmas. For simplicity we focus on a collection $\mathcal{S}$ of unit squares. In subsection 7.4 we discuss how to perform the extension to general rectangles. Specifically, in this section we provide our main lemmas concerning interactions between corner-chains of opposite corners and corner-chains of adjacent corners.

7.2.1. Corner-chains of adjacent corners. Consider a rectangle Q with side-lengths at most $1/2$. Let $\widetilde{\mathcal{S}}_{\urcorner} = \widetilde{\mathcal{S}}(Q,\urcorner)$ and $\widetilde{\mathcal{S}}_{\ulcorner} = \widetilde{\mathcal{S}}(Q,\ulcorner)$ denote corner-chains corresponding to adjacent corners $\urcorner$ and $\ulcorner$. (The other three cases of pairs of adjacent corners can be reduced to this case by "turning the picture.") We show that, by picking at most one square from each corner-chain, it is possible to "separate" between the chains. That is (as formalized in the next lemma), after picking one square from each chain, the squares having smaller indexes than those picked form a chain with respect to the remaining region.

Let $\{S_i\}_{i=1}^{m}$ (respectively, $\{S'_i\}_{i=1}^{m'}$) denote the ordering of the squares in $\widetilde{\mathcal{S}}_{\urcorner}$ (respectively, $\widetilde{\mathcal{S}}_{\ulcorner}$) in increasing (respectively, decreasing) order of the x-coordinate of their centers (or corners in Q). By Claim 7.3, both indexed sets $\widetilde{\mathcal{S}}_{\urcorner}$ and $\widetilde{\mathcal{S}}_{\ulcorner}$ are chains with respect to Q.

LEMMA 7.4. *There exist two squares, $S_k \in \widetilde{\mathcal{S}}_{\urcorner}$ and $S'_\ell \in \widetilde{\mathcal{S}}_{\ulcorner}$, such that*

1. *the prefixes $\{S_1,\ldots,S_{k-1}\}$ and $\{S'_1,\ldots,S'_{\ell-1}\}$ are disjoint; namely, for every $S_{k'}$ and $S'_{\ell'}$ such that $k' < k$ and $\ell' < \ell$, we have $S_{k'} \cap S'_{\ell'} = \emptyset$;*
2. *each of the prefixes $\{S_1,\ldots,S_{k-1}\}$ and $\{S'_1,\ldots,S'_{\ell-1}\}$ is a chain with respect to $Q \setminus (S_k \cup S'_\ell)$;*
3. *the union of S_k and S'_ℓ covers every point in Q that is covered by a square in one of the suffixes; namely, $(\bigcup_{t=k+1}^{m}(S_t \cap Q)) \bigcup (\bigcup_{t=\ell+1}^{m'}(S'_t \cap Q)) \subseteq S_k \cup S'_\ell$.*

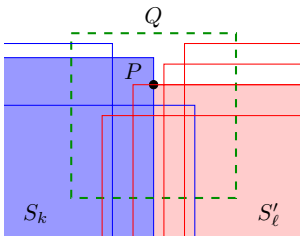


FIG. 7.5. *An illustration for Lemma 7.4. The region Q is depicted by a dashed rectangle. Only the corners of squares in the corner-chains are depicted. The two filled squares are the selected squares S_k and S'_ℓ.*

The implication of this lemma is that it is possible to select two squares S_k and S'_ℓ to serve all cells that are contained in the union $(\bigcup_{t=k}^{m}(S_t \cap Q)) \cup (\bigcup_{t=\ell}^{m'}(S'_t \cap Q))$. Furthermore, each of the prefixes is a chain with respect to the remaining region.

Proof. Consider the Q-envelopes of the two corner-chains. Both envelopes are "stairs" curves. By Claim 7.1, the Q-envelope of $\widetilde{\mathcal{S}}_\neg$ ($\widetilde{\mathcal{S}}_\ulcorner$) is nonincreasing (nondecreasing). Hence the Q-envelopes intersect at most once. If they do not intersect, then the claim is trivial (pick the last square from each chain). Otherwise, let P denote the intersection point. Let the selected squares S_k and S'_ℓ be the squares that intersect in point P. We assume that P is along the horizontal upper side of S'_ℓ (i.e., $P_y = y_\ulcorner(S'_\ell)$) and along the vertical right side of S_k (i.e., $P_x = x_\neg(S_k)$). (The reverse case is reduced to this case by "flipping the picture.")

Part 1 of the lemma follows by showing that the vertical line passing through P separates the prefixes. Namely, if $A \in S_{k'}$, $k' < k$, then $A_x < P_x$ (i.e., the x-coordinate of point A is less than the x-coordinate of point P). Similarly, if $B \in S'_{\ell'}$, $\ell' < \ell$, then $B_x > P_x$. Consider first any square $S_{k'} \in \widetilde{\mathcal{S}}_\neg$, where $k' < k$. By our assumption that P is along the vertical right side of S_k, we have that $P_x = x_\neg(S_k)$. By the ordering of the squares in $\widetilde{\mathcal{S}}_\neg$, we have that $x_\neg(S_{k'}) < x_\neg(S_k)$, and hence for every $A \in S_{k'}$, $A_x \leq x_\neg(S_{k'}) < x_\neg(S_k)$. It directly follows that $A_x < P_x$. Next consider a square $S'_{\ell'} \in \widetilde{\mathcal{S}}_\ulcorner$, where $\ell' < \ell$. By our assumption that P is along the horizontal upper side of S'_ℓ, and by the ordering of the squares in $\widetilde{\mathcal{S}}_\ulcorner$, necessarily $x_\ulcorner(S'_{\ell'}) > P_x$. But, for every point $B \in S'_{\ell'}$, $B_x \geq x_\ulcorner(S'_{\ell'})$, and so $B_x > P_x$.

To prove part 2 of the lemma it suffices to show that (i) $(S_{k'} \setminus S_k) \cap S'_\ell = \emptyset$ if $k' < k$, and (ii) $(S'_{\ell'} \setminus S'_\ell) \cap S_k = \emptyset$ if $\ell' < \ell$. This is sufficient since $\widetilde{\mathcal{S}}_\neg$ (respectively, $\widetilde{\mathcal{S}}_\ulcorner$) is a chain with respect to Q. Hence, every cell corresponding to an interval $[i,j] \subseteq [1,k-1]$ (respectively, $[i,j] \subseteq [1,\ell-1]$) of $\widetilde{\mathcal{S}}_\neg$ (respectively, $\widetilde{\mathcal{S}}_\ulcorner$) in Q is disjoint from $S_k \cup S'_\ell$, and the full interval property is preserved. For example, consider the two squares in Figure 7.5 that belong to the $\ulcorner$-chain and are above S'_ℓ. If we denote them by S_1 and S_2, then we see that the regions $S_1 \setminus S'_\ell$ and $S_2 \setminus S'_\ell$ are both disjoint from S_k and that $\{S_1, S_2\}$ form a chain with respect to $Q \setminus (S_k \cap S'_\ell)$.

In order to verify (i), consider a square $S_{k'}$ for $k' < k$. To show that $(S_{k'} \setminus S_k) \cap S'_\ell = \emptyset$, consider a point $A \in (S_{k'} \setminus S_k)$. The ordering of $\widetilde{\mathcal{S}}_\neg$ implies that $A_y > y_\neg(S_k)$, and by the definition of P, $y_\neg(S_k) \geq P_y$. Since $P_y = y_\ulcorner(S'_\ell)$, we get that A is above S'_ℓ and, in particular, $A \notin S'_\ell$ as claimed in (i). Item (ii) is proved analogously, and part 2 of the lemma follows.

It remains to prove part 3 of the lemma. Consider a point $A \in S_{k'} \cap Q$, for $k' > k$. There are two possibilities. (i) $A_x \leq P_x$: In this case, $A_y \leq y_\neg(S_{k'}) \leq y_\neg(S_k)$.

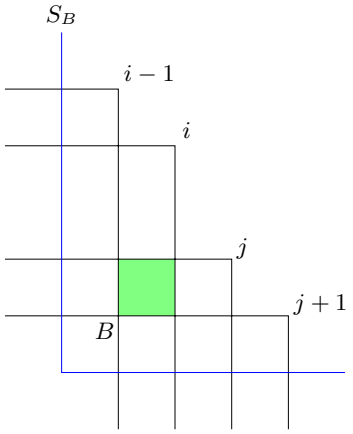


FIG. 7.6. *The construction in the proof of Lemma* 7.6.

Since $P_x = x_{\urcorner}(S_k)$, we get that $A \in S_k$. (ii) $A_x > P_x$: If $A_y \geq P_y$, then $\urcorner(S_{k'})$ is above and to the right of P, and hence $P \in S_{k'}$, a contradiction. It follows that $A_y < P_y = y_{\ulcorner}(S'_\ell)$. Since $P_x \geq x_{\ulcorner}(S'_\ell)$, we get that $A \in S'_\ell$. Therefore, the suffix of $\widetilde{\mathcal{S}}_{\urcorner}$ is covered by $S_k \cup S'_\ell$. The proof for the suffix of $\widetilde{\mathcal{S}}_{\ulcorner}$ is analogous, and part 3 of the lemma follows. □

7.2.2. Corner-chains of opposite corners. Consider a rectangle Q with side lengths at most 1/2. Let $\widetilde{\mathcal{S}}_{\urcorner} = \widetilde{\mathcal{S}}(Q, \urcorner)$ and $\widetilde{\mathcal{S}}_{\llcorner} = \widetilde{\mathcal{S}}(Q, \llcorner)$ denote corner-chains corresponding to opposite corners $\urcorner$ and $\llcorner$. (The case of the $\lrcorner$-corner and $\ulcorner$-corner is reduced to this case by "flipping or rotating the picture.") Let $Q_{\urcorner} = Q \cap \bigcup_{S \in \widetilde{\mathcal{S}}_{\urcorner}} S$ and $Q_{\llcorner} = Q \cap \bigcup_{S \in \widetilde{\mathcal{S}}_{\llcorner}} S$. Our goal is to select an approximately minimum subset from each corner-chain so as to cover $Q_{\urcorner} \cup Q_{\llcorner}$. To this end, we find minimal covers of $Q_{\urcorner} \setminus Q_{\llcorner}$, $Q_{\llcorner} \setminus Q_{\urcorner}$, and $Q_{\urcorner} \cap Q_{\llcorner}$.

DEFINITION 7.5. *A subset* $\widetilde{\mathcal{S}}^m_{\urcorner} \subseteq \widetilde{\mathcal{S}}_{\urcorner}$ *is a* minimal cover *of* $Q_{\urcorner} \setminus Q_{\llcorner}$ *if* (i) $\widetilde{\mathcal{S}}^m_{\urcorner}$ *covers* $Q_{\urcorner} \setminus Q_{\llcorner}$*, and* (ii) *no proper subset of* $\widetilde{\mathcal{S}}^m_{\urcorner}$ *covers* $Q_{\urcorner} \setminus Q_{\llcorner}$.

The following lemma shows that minimal covers of $(Q_{\urcorner} \setminus Q_{\llcorner})$ are chains with respect to $(Q_{\urcorner} \setminus Q_{\llcorner})$.

LEMMA 7.6. *If* $\widetilde{\mathcal{S}}^m_{\urcorner} \subseteq \widetilde{\mathcal{S}}_{\urcorner}$ *is a minimal cover of* $Q_{\urcorner} \setminus Q_{\llcorner}$*, and the squares in* $\widetilde{\mathcal{S}}^m_{\urcorner}$ *are indexed according to the* x*-coordinate of their* $\urcorner$*-corners, then* $\widetilde{\mathcal{S}}^m_{\urcorner}$ *is a chain with respect to* $Q_{\urcorner} \setminus Q_{\llcorner}$.

Proof. Let $\widetilde{\mathcal{S}}^m_{\urcorner} = \{S'_1, \ldots, S'_k\}$. Since $\widetilde{\mathcal{S}}_{\urcorner}$ is a chain with respect to Q, it follows that $\widetilde{\mathcal{S}}^m_{\urcorner}$ is also a chain with respect to $Q_{\urcorner}$. For simplicity, add "dummy" squares S'_0 and S'_{k+1} to $\widetilde{\mathcal{S}}^m_{\urcorner}$, where $\urcorner(S'_0) = \ulcorner(Q)$ and $\urcorner(S'_{k+1}) = \lrcorner(Q)$. Note that these dummy squares do not assist in covering $Q_{\urcorner} \setminus Q_{\llcorner}$. For the sake of contradiction, assume that $\widetilde{\mathcal{S}}^m_{\urcorner}$ is not a chain with respect to $Q_{\urcorner} \setminus Q_{\llcorner}$. Since $\widetilde{\mathcal{S}}^m_{\urcorner}$ is not a chain, it is not empty. Consider an interval $[i, j]$, for $0 < i \leq j < k+1$, such that the corresponding cell in $\mathcal{A}(\widetilde{\mathcal{S}}^m_{\urcorner})$ is contained in $Q_{\llcorner}$. (See Figure 7.6 for an illustration of this case.) Consider the corner B of the cell $[i, j]$ in Q defined by the intersection of the sides of S'_{i-1} and S'_{j+1} in Q. Since the cell $[i, j]$ is in $Q_{\llcorner}$, so is the point B. Let $S_B \in \widetilde{\mathcal{S}}_{\llcorner}$ denote a square that contains B. It follows that the whole cell $[i, j]$ as well as $\urcorner(S'_{i-1})$, $\urcorner(S'_i)$, $\urcorner(S'_j)$, and $\urcorner(S'_{j+1})$ are in S_B. It is easy to see that we may omit both S'_i and S'_j from $\widetilde{\mathcal{S}}^m_{\urcorner}$ while still covering $Q_{\urcorner} \setminus Q_{\llcorner}$, contradicting the assumption that $\widetilde{\mathcal{S}}^m_{\urcorner}$ is a minimal

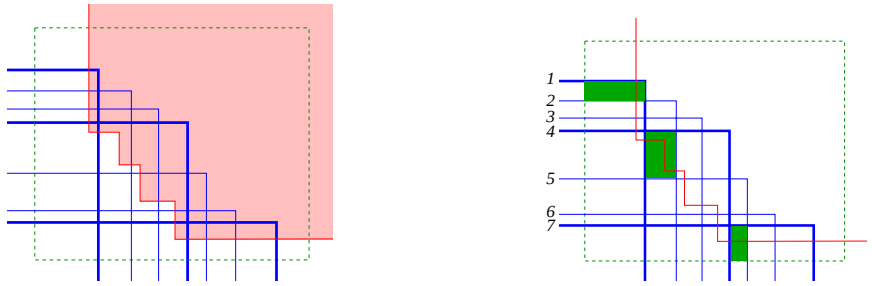


FIG. 7.7. *A minimal cover of the union of opposite corner-chains.* (A) *The rectangle Q is depicted by a dashed rectangle. The union of the "upper" corner-chain $Q_{\llcorner}$ is shaded (so that $Q_{\urcorner} \setminus Q_{\llcorner}$ is the unshaded region within Q). A minimal cover $\widetilde{\mathcal{S}}^m_{\urcorner} \subseteq \widetilde{\mathcal{S}}_{\urcorner}$ is depicted by thick $\urcorner$-corners.* (B) *An explanation of how $\widetilde{\mathcal{S}}^m_{\urcorner}$ is greedily computed. Here only the boundary of $Q_{\llcorner}$ is depicted. Cells $[1,1]$, $[2,4]$, and $[5,7]$ are shaded. Since cell $[1,1]$ is not completely covered by $Q_{\llcorner}$, we add square 1 to the cover. Cells $[2,2]$ and $[2,3]$ are covered by $Q_{\llcorner}$, but cell $[2,4]$ is not; therefore, square 4 is added to the cover. Now cells $[5,5]$ and $[5,6]$ are covered by $Q_{\llcorner}$, but cell $[5,7]$ is not; therefore, square 7 is added to the minimal cover.*

cover. □

An algorithm for finding a minimal cover. We now describe a greedy algorithm for finding a subset $\widetilde{\mathcal{S}}^m_{\urcorner} \subseteq \widetilde{\mathcal{S}}_{\urcorner}$ that is a minimal cover of $Q_{\urcorner} \setminus Q_{\llcorner}$. Let $S_1, \ldots, S_m$ be an ordering of the squares in $\widetilde{\mathcal{S}}_{\urcorner}$ according to the increasing value of $x_{\urcorner}(S_i)$. Recall that $\widetilde{\mathcal{S}}_{\urcorner}$ is a chain with respect to Q, and therefore every subset of $\widetilde{\mathcal{S}}_{\urcorner}$ is a chain with respect to Q. For any two indexes $1 \leq a \leq b \leq m$, let $\widetilde{\mathcal{S}}_{\urcorner}[a,b]$ denote the cell v in the arrangement $\mathcal{A}(\widetilde{\mathcal{S}}_{\urcorner})$ such that $N(v) = \{S_a, \ldots, S_b\}$.

The greedy algorithm works in an iterative fashion. Let k be the index of the square selected in the last iteration (where initially $k = 0$ and $\widetilde{\mathcal{S}}^m_{\urcorner} = \emptyset$). Consider all cells $\widetilde{\mathcal{S}}_{\urcorner}[k+1, \ell]$, where $(k+1) \leq \ell \leq m$, such that $\widetilde{\mathcal{S}}_{\urcorner}[k+1, \ell] \cap Q$ is not fully contained in $Q_{\llcorner}$. If there is no such cell, then the algorithm terminates. Otherwise, let ℓ be the minimum index such that $\widetilde{\mathcal{S}}_{\urcorner}[k+1, \ell]$ is not fully contained in $Q_{\llcorner}$, and add S_ℓ to $\widetilde{\mathcal{S}}^m_{\urcorner}$. For an example, see Figure 7.7(B).

CLAIM 7.7. *The greedy algorithm computes a minimal cover $\widetilde{\mathcal{S}}^m_{\urcorner} \subseteq \widetilde{\mathcal{S}}_{\urcorner}$ of $Q_{\urcorner} \setminus Q_{\llcorner}$.*

By "rotating the picture," we can obtain an analogous claim concerning a minimal cover $\widetilde{\mathcal{S}}^m_{\llcorner} \subseteq \widetilde{\mathcal{S}}_{\llcorner}$ of $Q_{\llcorner} \setminus Q_{\urcorner}$.

Proof. Let $k_1 < k_2 < \cdots < k_r$ denote the sequence of squares added to $\widetilde{\mathcal{S}}^m_{\urcorner}$ by the greedy algorithm. We show that the algorithm computes a cover $\widetilde{\mathcal{S}}^m_{\urcorner}$ of $Q_{\urcorner} \setminus Q_{\llcorner}$, by showing that the following invariant holds throughout the algorithm:

$$(Q_{\urcorner} \setminus Q_{\llcorner}) \cap \bigcup_{i=1}^{k_t} S_i \subseteq \bigcup_{j=1}^{t} S_{k_j}.$$

The invariant holds trivially when the algorithms starts (as $k_t = 0$). Assume, for the sake of contradiction, that a cell $\widetilde{\mathcal{S}}_{\urcorner}[i,j]$ (for $i \leq j < k_t$) in $Q_{\urcorner} \setminus Q_{\llcorner}$ is not covered by $\bigcup_{j \leq t} S_{k_j}$. If $i \leq k_{t-1}$, then there are two cases: (i) $j \leq k_{t-1}$, in which case the induction hypothesis already implies that cell $\widetilde{\mathcal{S}}_{\urcorner}[i,j]$ is contained in $\bigcup_{j<t} S_{k_j}$, and (ii) $j > k_{t-1}$, in which case cell $\widetilde{\mathcal{S}}_{\urcorner}[i,j]$ is contained in $S_{k_{t-1}}$. Both cases lead to a contradiction, so we assume that $i > k_{t-1}$. It can be verified that if the cell $\widetilde{\mathcal{S}}_{\urcorner}[i,j]$

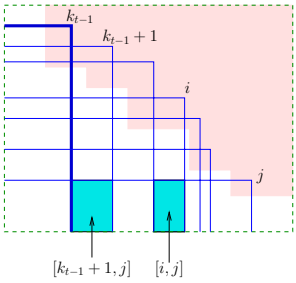


FIG. 7.8. *An illustration for the case $i > k_{t-1}$ in the proof of Claim 7.7.*

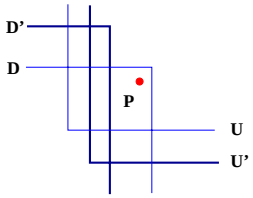


FIG. 7.9. *An illustration for Lemma 7.8. The point $P \in Q_\neg \cap Q_\llcorner$ is in $D \cap U$, where neither $D \in \widetilde{\mathcal{S}}^m_\neg$ nor $U \in \widetilde{\mathcal{S}}^m_\llcorner$. Here $D' \in \widetilde{\mathcal{S}}^m_\neg$ (the square that covers the cell that contains the point slightly to the left of $\llcorner(U)$) is such that $y_\neg(D') > P_y$. Finally, $U' = B_\llcorner(D')$, that is, it is the last square from $\widetilde{\mathcal{S}}_\llcorner$ that intersects D'.*

is in $Q_\neg \setminus Q_\llcorner$, then the cell $\widetilde{\mathcal{S}}_\neg[k_{t-1}+1, j]$ is also in $Q_\neg \setminus Q_\llcorner$, but in such a case, the greedy algorithm would have chosen S_{k_t} such that $k_{t-1}+1 \le k_t \le j$, a contradiction. For an illustration of this case, see Figure 7.8.

The stopping condition of the algorithm, combined with the invariant, guarantees that, when the algorithm terminates, $\widetilde{\mathcal{S}}^m_\neg$ covers $Q_\neg \setminus Q_\llcorner$.

Minimality of $\widetilde{\mathcal{S}}^m_\neg$ is proved as follows. Consider a square $S_j \in \widetilde{\mathcal{S}}^m_\neg$. When S_j was added to $\widetilde{\mathcal{S}}^m_\neg$, it was added due to a cell $[i, j]$, with i greater than the index of the square added to $\widetilde{\mathcal{S}}^m_\neg$ just before S_j. The cell $[i, j]$ is covered only by S_j (among the squares in $\widetilde{\mathcal{S}}^m_\neg$), and hence minimality follows. □

Let $m_\neg = |\widetilde{\mathcal{S}}^m_\neg|$, and let $m_\llcorner = |\widetilde{\mathcal{S}}^m_\llcorner|$. Let $m = \max\{m_\neg, m_\llcorner\}$. In the next lemma we show that it is possible to cover $Q_\neg \cup Q_\llcorner$ by $O(m)$ squares from $\widetilde{\mathcal{S}}_\neg \cup \widetilde{\mathcal{S}}_\llcorner$.

LEMMA 7.8. *There exists a subset $\mathcal{S}' \subseteq \widetilde{\mathcal{S}}_\neg \cup \widetilde{\mathcal{S}}_\llcorner$ of $O(m)$ squares that covers $Q_\neg \cup Q_\llcorner$.*

Proof. Since $\widetilde{\mathcal{S}}^m_\neg$ (respectively, $\widetilde{\mathcal{S}}^m_\llcorner$) covers $Q_\neg \setminus Q_\llcorner$ (respectively, $Q_\llcorner \setminus Q_\neg$) and $|\widetilde{\mathcal{S}}^m_\neg \cup \widetilde{\mathcal{S}}^m_\llcorner| \le 2m$, the remaining problem is to cover $Q_\neg \cap Q_\llcorner$ using $O(m)$ squares. For every square $S \in \widetilde{\mathcal{S}}^m_\neg$ consider the set $\widetilde{\mathcal{S}}_\llcorner(S)$ of squares in $\widetilde{\mathcal{S}}_\llcorner$ that intersect S. Define $A_\llcorner(S)$ (respectively, $B_\llcorner(S)$) to be the first (respectively, last) square in $\widetilde{\mathcal{S}}_\llcorner(S)$ when sorted according to the y-coordinates of their $\llcorner$-corners. We claim that $\bigcup_{S \in \widetilde{\mathcal{S}}^m_\neg}(A_\llcorner(S) \cup B_\llcorner(S))$ covers $(Q_\neg \cap Q_\llcorner) \setminus (\widetilde{\mathcal{S}}^m_\neg \cup \widetilde{\mathcal{S}}^m_\llcorner)$. Hence, we need to add at most two squares from $\widetilde{\mathcal{S}}_\llcorner$ per square in $\widetilde{\mathcal{S}}^m_\neg$ to cover $(Q_\neg \cap Q_\llcorner) \setminus (\widetilde{\mathcal{S}}^m_\neg \cup \widetilde{\mathcal{S}}^m_\llcorner)$.

Consider a point $P \in Q_\neg \cap Q_\llcorner$. Let $D \in \widetilde{\mathcal{S}}_\neg$ (respectively, $U \in \widetilde{\mathcal{S}}_\llcorner$) denote a square that contains P. If $D \in \widetilde{\mathcal{S}}^m_\neg$ or $U \in \widetilde{\mathcal{S}}^m_\llcorner$, then we are done. Otherwise, consider the cell in $\mathcal{A}(\widetilde{\mathcal{S}}_\neg)$ that contains a point slightly to the left of $\llcorner(U)$. This cell is in $Q_\neg \setminus Q_\llcorner$, and therefore there exists a square $D' \in \widetilde{\mathcal{S}}^m_\neg$ that covers this

cell. If $P \in D'$, we are done. Otherwise, we consider two cases: $y_{\neg}(D') \geq P_y$ and $y_{\neg}(D') < P_y$. In the first case consider the square $U' = B_{\llcorner}(D')$. Such a square exists since U intersects D'. We can now bound the coordinates of $_{\llcorner}(U')$ to show that $P \in U'$ as follows: (i) $x_{\llcorner}(U') \leq x_{\neg}(D') < P_x$ (the first inequality holds because U' and D' intersect, and the second inequality holds because $y_{\neg}(D') \geq P_y$ while $P \notin D'$); (ii) $y_{\llcorner}(U') \leq y_{\llcorner}(U) \leq P_y$ (the first inequality holds because $U' = B_{\llcorner}(D')$, and the second by the premise of this case). Thus $P \in U'$. For an illustration of this case, see Figure 7.9. The second case in which $y_{\neg}(D') < P_y$ is treated analogously, where here we let $U' = A_{\llcorner}(D')$. The claim follows. □

Remark 1. Lemmas 7.6 and 7.8 and Claim 7.7 regarding opposite corner-chains were stated with respect to a rectangle Q that is contained in a tile. The same lemmas and claim hold with respect to a region $Q \subseteq T$ that satisfies the following properties.

The region Q contains two designated points $C_{\llcorner}$ and $C_{\neg}$. (When Q is a rectangle, then $C_{\llcorner}$ is the bottom-left corner and $C_{\neg}$ is the top-right corner.) The point $C_{\llcorner}$ is contained in every square in $\widetilde{\mathcal{S}}_{\neg}$, and the point $C_{\neg}$ is contained in every square in $\widetilde{\mathcal{S}}_{\llcorner}$. Moreover, if a square $S \in \widetilde{\mathcal{S}}_{\neg}$ (respectively, $S \in \widetilde{\mathcal{S}}_{\llcorner}$) contains the point $C_{\neg}$ (respectively, $C_{\llcorner}$), then $Q_{\llcorner} \setminus Q_{\neg} = \emptyset$ (respectively, $Q_{\llcorner} \setminus Q_{\neg} = \emptyset$).

As we discuss in more detail shortly, if Lemma 7.4 is applied to separate corner-chains of adjacent corners, then the remaining uncovered region in a tile is a region that satisfies the above condition. Hence, after separating corner-chains of adjacent corners, we may apply Claim 7.7 and Lemma 7.8 for the covering of the remaining region in the tile.

Remark 2. After the separation of adjacent corner-chains in a tile, it is not possible for both pairs of opposite corner-chains to intersect. Namely, at most one pair of opposite corner-chains may intersect. We do not rely on this property, the proof of which is easy.

7.3. Coloring arrangements of squares. In this section we prove Theorem 1.4 for unit squares. The goal of the algorithm is to pick an "essential" subset of squares per tile whose union must be served. The coloring of the essential squares per tile is done according to Theorem 1.5. Recall that a tile is an orphan tile if it does not contain a center of a square. As noted at the start of this section, the main thrust of the algorithm and its analysis is in serving the covered regions in orphan tiles (i.e., the union of the squares minus the union of nonorphan tiles). The task of selecting a subset of squares that serves the covered parts of orphan tiles is "done independently" by the orphan tiles. The set of essential squares per nonorphan tile is the set of squares that belong to the tile and have been selected by one of the neighboring orphan tiles.

7.3.1. Selection of squares by nonbare orphan tiles. Consider a nonbare orphan tile T. In this section we describe how squares from neighboring tiles are selected by T so that these squares serve the area that is the intersection of T with their union.

Selection of squares consists of three steps: (1) selection of at most one square from each e-neighbor—this step maximizes service from e-neighbors; (2) selection of at most two squares from each v-neighbor—this step resolves all interactions between chains of squares corresponding to adjacent corners; (3) final selection of squares from the remaining chains corresponding to corners—this step takes into account interactions between chains corresponding to opposite corners.

Selecting squares from e-neighbors. Consider the tile T and the set of squares that

belong to an e-neighbor T^e of T. For brevity, assume that T^e is to the left of T and that $\mathcal{S}(T^e) \neq \emptyset$. Every square $S \in \mathcal{S}(T^e)$ covers a vertical strip of T. If we select the rightmost square S in $\mathcal{S}(T^e)$, then we get that for every $S' \in \mathcal{S}(T^e)$, $S' \cap T \subseteq S \cap T$. In the same fashion, we select the closest square to T from each e-neighbor of T. By selecting at most one square from each e-neighbor of T, the first substep covers all the points in $T \cap \bigcup_{T^e \in e\text{-neighbors}(T)} \mathcal{S}(T^e)$.

After this step, the region within the tile T that still needs to be served is a rectangle. Let us denote this rectangle by T'. Note that the union of squares in $\mathcal{S}$ may either fully cover or partly cover the rectangle T'. In any case, only squares that belong to v-neighbors of T intersect T'. An illustration of this step was provided in Figure 7.2.

Selecting squares from v-neighbors: Adjacent corners. Consider the rectangle $T' \subseteq T$ and a corner γ. The squares of $\mathcal{S}(T', \gamma)$ that participate in the T'-envelope are denoted by $\widetilde{\mathcal{S}}(T', \gamma)$. By Claim 7.3, $\widetilde{\mathcal{S}}(T', \gamma)$ is a chain with respect to T' when indexed according to the x-coordinate of its centers (or γ-corners). By applying Lemma 7.4 to the four appropriate pairs of chains corresponding to adjacent corners, we obtain at most eight squares that serve as "separators" between the pairs of chains. The selected squares cover all points in T' that are covered by squares in the tails of the chains. Each corner-chain is reduced to a consecutive block of squares between the two selected squares in that chain. The remaining portions of adjacent corner-chains are disjoint. Let T'' denote the subregion consisting of T' minus the union of the (at most eight) selected squares. By our notational convention, $\widetilde{\mathcal{S}}(T'', \gamma)$ denotes the subset of squares in $\mathcal{S}(T'', \gamma)$ that participate in the T'' envelope. Note that, by Lemma 7.4, $\widetilde{\mathcal{S}}(T'', \gamma)$ is a chain with respect to T''.

Selecting squares from v-neighbors: Opposite corners. In the third step we apply Claim 7.7 and Lemma 7.8 to each pair of subsets $\widetilde{\mathcal{S}}(T'', \gamma)$ and $\widetilde{\mathcal{S}}(T'', \mathrm{op}(\gamma))$. This application determines the subsets of $\widetilde{\mathcal{S}}(T'', \gamma)$ (and $\widetilde{\mathcal{S}}(T'', \mathrm{op}(\gamma))$) that suffice to serve the intersection of T'' with the union of each pair of chains. Note that, due to the separation of adjacent corner-chains (see Lemma 7.4), at most one pair of opposite corner-chains may intersect.

A subtle issue to be addressed is whether the remaining region $T'' \subseteq T' \subseteq T$ in the beginning of this step satisfies the premises of Remark 1 for each pair of opposite corner-chains. Consider, for example, the subset $\widetilde{\mathcal{S}}(T'', \urcorner)$. This subset is a consecutive block of squares from $\widetilde{\mathcal{S}}(T', \urcorner)$. Let $S^f_{\urcorner}$ and $S^{\ell}_{\urcorner}$ denote the squares in $\widetilde{\mathcal{S}}(T', \urcorner) \setminus \widetilde{\mathcal{S}}(T'', \urcorner)$ that "hug" this block (i.e., $S^f_{\urcorner}$ and $S^{\ell}_{\urcorner}$ were selected in the adjacent corner-chain stage). The designated point $C_{\llcorner} \in T''$ is the intersection of the right side of $S^f_{\urcorner}$ and the top side of $S^{\ell}_{\urcorner}$. One can define in this fashion all four designated points C_γ for $\gamma \in \Gamma$. We can now apply Lemma 7.8 to each pair of subsets $\widetilde{\mathcal{S}}(T'', \gamma)$ and $\widetilde{\mathcal{S}}(T'', \mathrm{op}(\gamma))$, where the corresponding designated points that satisfy the premise of Remark 1 are $C_{\mathrm{op}(\gamma)}$ and C_γ. For an illustration, see Figure 7.10.

7.3.2. Coloring the essential squares. In the previous steps, each orphan tile T_o selected a subset of squares used to serve the points in $T_o \cap \bigcup \mathcal{S}$. Given a nonorphan tile T, let $sel(T) \subseteq \mathcal{S}(T)$ denote the subset of squares whose centers reside in T that are selected by some tile in order to participate in its cover. If no square in $\mathcal{S}(T)$ is requested from orphan tiles, then we select an arbitrary square in $\mathcal{S}(T)$ to serve T, and let $sel(T)$ contain only this square. At this stage we apply Theorem 1.5 and color each subset $sel(T)$ by $O(\log |sel(T)|)$ colors; these colors are taken from the palette assigned to the tile T.

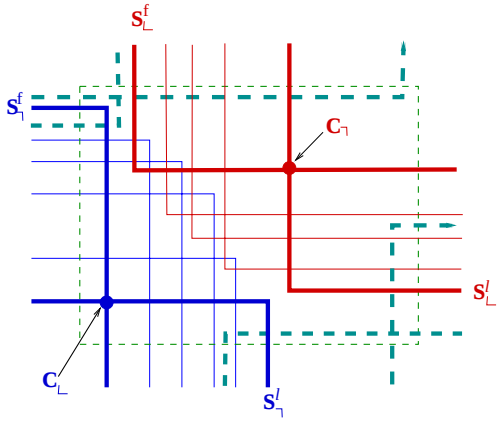


FIG. 7.10. *An illustration for the choice of points C_γ and $C_{\mathrm{op}(\gamma)}$ that satisfy the premise of Remark* 1. *The selected squares, which determine the two points, are labeled and marked in bold. The dashed bold corners of squares correspond to the four other selected squares that belong to the adjacent chains $\widetilde{\mathcal{S}}(T', \ulcorner)$ and $\widetilde{\mathcal{S}}(T', \lrcorner)$. The thin dashed rectangle is T', and T'' is the region obtained by removing the four bold and four dashed-bold squares from T'.*

Recall that at most one square from $\mathcal{S}(T)$ was requested from each of its four e-neighbors. Each of its four v-neighbors initially requested at most two squares (as "separators" between chains). These requests amount to at most twelve squares. The main contribution to $sel(T)$ is due to the subsets of squares that were requested by v-neighbors of T in the last selection step, that is, in the step that deals with opposite corner-chains.

For each corner type γ, let $sel_\gamma(T)$ denote the subset of squares in $sel(T)$ that were selected by $T_{\mathrm{op}(\gamma)}$ in the opposite-corners selection step. The γ-corners of squares in $sel_\gamma(T)$ are contained in the tile $T_{\mathrm{op}(\gamma)}$. Let $m_\gamma(T) = |sel_\gamma(T)|$. Since $|sel(T)| = O(\max_{\gamma\in\Gamma}\{m_\gamma(T)\})$, and since there are nine palettes, the next corollary follows.

COROLLARY 7.9. *For any given set of unit squares $\mathcal{S}$, it is possible to CF-color $\mathcal{S}$ using $O\left(\log\left(\max_{T,\gamma}\{m_\gamma(T)\}\right)\right)$ colors.*

7.3.3. A lower bound for optimal CF-coloring. In this section we lower-bound the number of colors required by an optimal CF-coloring. Recall that, for a tile T and corner $\gamma \in \Gamma$, the set of squares that intersect T with corner type γ is denoted by $\mathcal{S}(T,\gamma)$. Recall that $\widetilde{\mathcal{S}}(T,\gamma)$ denotes the subset of squares from $\mathcal{S}(T,\gamma)$ that appear in the T-envelope.

Let T be any (orphan) tile, and let γ be a corner. The following lemma states a lower bound on $\chi_{\mathrm{opt}}(\mathcal{S})$ in terms of the size of a subset $\mathcal{S}' \subseteq \widetilde{\mathcal{S}}(T,\gamma)$ that is a chain with respect to the region $Q = T \setminus \{S : S \notin \mathcal{S}(T,\gamma)\}$. That is, the region Q is what remains of T after we remove all squares that intersect T with the exception of squares in $\mathcal{S}(T,\gamma)$.

LEMMA 7.10. *Let T denote a tile, γ a corner type, and let $Q = T \setminus \{S : S \notin \mathcal{S}(T,\gamma)\}$. Let $\mathcal{S}' \subseteq \widetilde{\mathcal{S}}(T,\gamma)$ be a chain with respect to Q. Then every CF-coloring of $\mathcal{A}(\mathcal{S})$ requires $\Omega(\log|\mathcal{S}'|)$ colors.*

The proof of Lemma 7.10 follows the same outline as the proof of Lemma 6.3. In fact, the same lower bound holds also for CF-multicoloring, implying Theorem 9.2 (see section 9). The only difference is that here we need to take into account that squares from $\mathcal{S}(T,\gamma) \setminus \mathcal{S}'$ may cover points in Q and hence can potentially serve cells

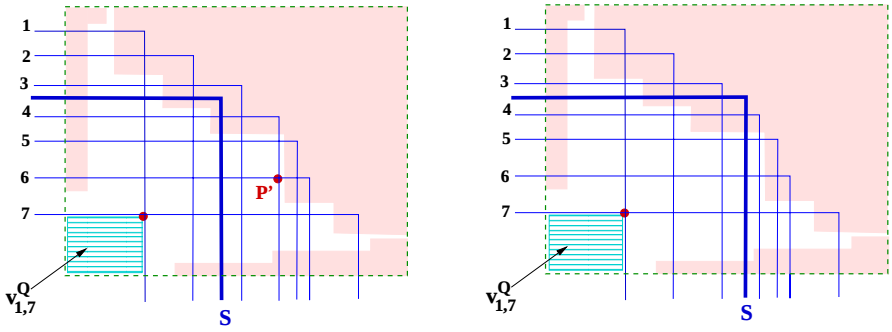


FIG. 7.11. *Illustration for the proof of Lemma* 7.10. *The squares of* $\mathcal{S}'$ *are labeled from* 1 *to* 7. *The region* Q *is the nonshaded region within the dashed rectangle. The point* P *is the top-right corner of* $v_{1,7}$ *(which here equals* $v^Q_{1,7}$*). On the left is an illustration of the case in which* P *is served (in an optimal CF coloring) by a square* S *such that* $S \cap Q \subseteq S_3 \cap Q$. *The point* P' *in this case is the top-right corner in a cell of the chain determined by* $S_4, \ldots, S_7$. *On the right is an illustration of the case in which* P *is served by a square* S *whose top-right corner does not reside in any square of* $\mathcal{S}'$.

in the chain $\mathcal{S}'$. For an illustration of the proof of the lemma, see Figure 7.11.

Proof. We consider the case $\gamma = \urcorner$. All other cases are proved analogously. Let $S_1, \ldots, S_m$ be an ordering of the squares in $\mathcal{S}'$ so that $x_\urcorner(S_1) < \cdots < x_\urcorner(S_m)$. For every $1 \le i \le j \le m$, let $v_{i,j}$ denote the cell in the arrangement $\mathcal{A}(\mathcal{S}')$ for which $N(v_{i,j}) = \{S_i, \ldots, S_j\}$. (Recall that for a cell v, $N(v)$ denotes the subset of regions that contain v.) Let $v^Q_{i,j} = v_{i,j} \cap Q$, where we know that $v^Q_{i,j}$ is nonempty for every $1 \le i \le j \le m$ because $\mathcal{S}'$ is a chain with respect to Q. Since we are dealing with squares (or, more generally, rectangles), we know that each $v_{i,j}$ is a rectangle. However, it may be the case that $v^Q_{i,j}$ is not a rectangle. The exact structure of $v^Q_{i,j}$ is actually immaterial to the proof. What will be needed is the following subclaim (where we assume that $m > 3$ or else Lemma 7.10 holds trivially).

Subclaim. For every $1 \le i, j \le m$ such that $i \le j - 2$, the top-right corner of $v_{i,j}$ is contained also in $v^Q_{i,j}$.

Proof of subclaim. Assume that the subclaim does not hold. This means that there exists some square $S' \notin \mathcal{S}(T, \urcorner)$ that contains the top-right corner of $v_{i,j}$. But in such a case either $S' \supseteq v_{i+1,j}$ or $S' \supseteq v_{i+1,j-1}$ or $S' \supseteq v_{i,j-1}$, contradicting the premise of the lemma that $\mathcal{S}'$ is a chain with respect to Q. The subclaim is thus established. For an illustration, see Figure 7.12.

Now consider an optimal CF-coloring χ_{opt} of $\mathcal{S}$. Let P be the top-right corner of $v_{1,m}$. By the above subclaim (assuming $m > 3$), $P \in v^Q_{1,m}$. Since $P \in Q$, only squares in $\mathcal{S}(T, \urcorner)$ contain P. Let $S \in \mathcal{S}(T, \urcorner)$ be a square that serves P in the coloring χ_{opt}. We first consider the case that $\urcorner(S)$, the top-right corner of S, is contained in some $S_k \in \mathcal{S}'$ (in particular, S_k may equal S). In this case, $S \cap Q \subseteq S_k \cap Q$.

We make two observations. The first is that both $\{S_1, \ldots, S_{k-1}\}$ and $\{S_{k+1}, \ldots, S_m\}$ are chains with respect to $Q \setminus S$. This is true since $\{S_1, \ldots, S_m\}$ is a chain with respect to Q, (hence both subsets must be chains with respect to $Q \setminus S_k$), and $Q \cap S \subseteq Q \cap S_k$. Thus, S cannot fully serve any of the cells in these two subchains. The second observation is that every square $S' \in \mathcal{S}(T, \urcorner)$ that serves the top-right corner P' of a cell in one of these chains (that corresponds to an interval of size at least 3) must also contain P. This is true because every top-right corner of such a

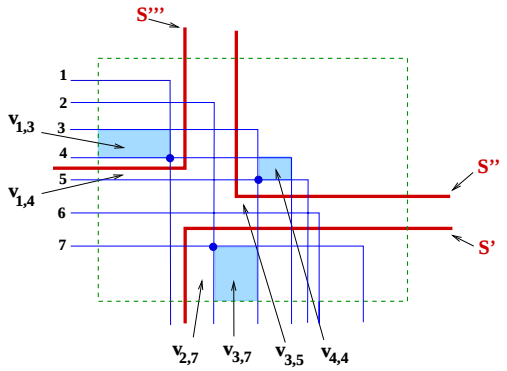


FIG. 7.12. *An illustration for the proof of the subclaim in the proof of Lemma* 7.10: (i) $i = 2$, $j = 7$, *and* S' *covers* $v_{3,7}$; (ii) $i = 3$, $j = 5$, *and* S'' *covers* $v_{4,4}$; *and* (iii) $i = 1$, $j = 4$, *and* S''' *covers* $v_{1,3}$.

cell dominates P (i.e., the x and y coordinates of such corners are not smaller than P_x and P_y, respectively). Since S serves P, it follows that the color of every square that contains P must be different from $\chi_{\text{opt}}(S)$.

If $\urcorner(S)$, the top-right corner of S, is not contained in any $S_k \in \mathcal{S}' = \{S_1, \ldots, S_m\}$, then define k as follows: $k = \max\{i : x_\urcorner(S_i) < x_\urcorner(S)\}$. Since $\mathcal{S}'$ is a chain with respect to Q, it follows that $\{S_1, \ldots, S_k\}$ and $\{S_{k+1}, \ldots, S_m\}$ are chains with respect to $Q \setminus S$. Furthermore, similarly to what was shown above, for any square $S' \in \mathcal{S}(T, \urcorner)$ that can serve the top-right corner of a cell in one of these chains (that corresponds to an interval of size at least 3) $\chi_{\text{opt}}(S') \neq \chi_{\text{opt}}(S)$. In either case we get the recurrence relation

$$|\chi_{\text{opt}}(\{S_1, \ldots, S_m\})| \geq 1 + \min_{1 \leq k \leq m} \{\max\{|\chi_{\text{opt}}(\{S_1, \ldots, S_{k-1}\})|, |\chi_{\text{opt}}(\{S_{k+1}, \ldots, S_m\})|\}\},$$

where for any subset $\mathcal{S}''$ of less than three squares, $|\chi_{\text{opt}}(\mathcal{S}'')| \geq 1$. Hence $|\chi_{\text{opt}}(\mathcal{S})| = \Omega(\log |\mathcal{S}'|)$, and the lemma follows. □

For any nonorphan tile T and corner γ, let $sel_\gamma(T)$ and $m_\gamma(T)$ $(= |sel_\gamma(T)|)$ be as defined preceding Corollary 7.9. Using Lemma 7.10, we establish the following lower bound.

LEMMA 7.11. *For any given set of unit squares* $\mathcal{S}$, *we have that* $|\chi_{\text{opt}}(\mathcal{S})| = \Omega\left(\log\left(\max_{T,\gamma}\{m_\gamma(T)\}\right)\right)$.

Proof. Consider a tile T and a corner type γ such that $m_\gamma(T)$ is maximal. Let W denote the tile $T_{\text{op}(\gamma)}$, which selected the squares in $sel_\gamma(T)$. Let Q denote the region remaining in W when the γ-corner chain and $\text{op}(\gamma)$-corner chain are considered in W. Assume, without loss of generality, that $\gamma = \urcorner$. The set $sel_\urcorner(T)$ consists of two kinds of squares: (i) squares that belong to the a minimal cover $\widetilde{\mathcal{S}}^m_\urcorner$ of $Q_\urcorner \setminus Q_\llcorner$ (these squares are selected by the greedy algorithm) and (ii) pairs of squares that were selected according to the procedure defined in Lemma 7.8. Let m' denote the number of squares of the first kind, and let m'' denote the number of squares of the second kind.

We consider first the case that $m' \geq m''$. The separation procedure of adjacent corner-chains combined with Lemma 7.6 implies that $\widetilde{\mathcal{S}}^m_\urcorner$ is a chain with respect to $W \setminus \{S : S \notin \mathcal{S}(W, \urcorner)\}$. By Lemma 7.10 we get that the number of required colors is

$\Omega(\log m') = \Omega(m_{\neg}(T))$.

We next consider the case that $m'' > m'$. Let V denote the tile $W_{\llcorner}$. The squares in $sel_{\llcorner}(V)$ were also selected by the tile W. The number of squares in $sel_{\llcorner}(V)$ that belong to a minimal cover $\widetilde{\mathcal{S}}^m_{\llcorner}$ of $Q_{\llcorner} \setminus Q_{\neg}$ is at least $m''/2$. By applying the same argument now on the tile V and the $\llcorner$-corner, the lemma follows. □

7.3.4. Wrapping up the proof of Theorem 1.4 for unit squares. Combining Corollary 7.9 and Lemma 7.11, and noting that the computational complexity of the algorithm is due only to sorting squares according their coordinates, Theorem 1.4 for unit squares directly follows.

7.4. General rectangles. Consider a collection $\mathcal{R}$ of rectangles with size-ratio ρ. Our goal is to prove the existence of an efficient algorithm for CF-coloring $\mathcal{R}$ that uses $O((\log \rho)^2) \cdot |\chi_{\text{opt}}(\mathcal{R})|$ colors. This means that if the size-ratio ρ is constant, then the algorithm is a constant-ratio approximation algorithm.

By separately scaling the x-axis and the y-axis, we may assume that the minimum width and height of rectangles in $\mathcal{R}$ are equal to 1. Hence, all side-lengths are in the range $[1, \rho]$.

The algorithm proceeds in two steps (as in the proof of Theorem 1.2). First, consider the case of $\rho \leq 2$. For this case we show that $O(|\chi_{\text{opt}}(\mathcal{R})|)$ colors suffice. For the more general case of $\rho > 2$, we partition the set of rectangles into $\log^2 \rho$ classes. For $1 \leq i, j < \log \rho + 1$, the class $\mathcal{R}^{i,j}$ consists of rectangles whose width is in the interval $[2^{i-1}, 2^i)$ and whose height is in the interval $[2^{j-1}, 2^j)$. Each class is colored using a distinct palette, to obtain a CF-coloring that uses $O((\log \rho + 1)^2) \cdot |\chi_{\text{opt}}(\mathcal{R})|$ colors, as required.

7.4.1. Rectangles with $\rho \leq 2$. We outline the algorithm for the case $\rho \leq 2$ below.

1. The tiling is the same as in the case of unit squares. The tiles are assigned 25 different palettes (instead of nine).
2. An orphan tile may now be completely covered by a rectangle. An orphan tile that is completely covered by a rectangle selects such a rectangle (this type of selection does not exist in the case of unit squares).
3. Instead of selecting closest rectangles from e-neighbors, every nonbare orphan tile that is not completely covered by a single rectangle selects the rightmost rectangle (if any) whose right edge intersects both the bottom and top side of the tile. The same selection takes place in the other three axis-parallel directions. In this stage an orphan cell selects at most four rectangles.
4. A nonbare orphan tile that still contains a region covered by $\mathcal{R}$ but not by the rectangles selected so far selects rectangles from the corner-chains as in the algorithm for unit squares. The reason that the same techniques apply is that the intersection of a rectangle with a tile contains at most one corner.
5. The essential (selected) rectangles from each tile are colored as described in the following paragraph.

Coloring the essential rectangles. Given a nonorphan tile T, let $sel(T)$ denote the set of rectangles that belong to T and were selected in the previous stages. For a corner type γ and a nonbare orphan tile W, let $\mathcal{R}'(W, \gamma)$ be the subset of rectangles selected by W whose γ-corner resides in W. Finally, let $m = \max_{W,\gamma}\{|\mathcal{R}'(W, \gamma)|\}$ denote the maximum (over all tiles W and corner types γ) of the number of rectangles selected by an orphan tile W due to their participation in a γ-corner-chain within the tile.

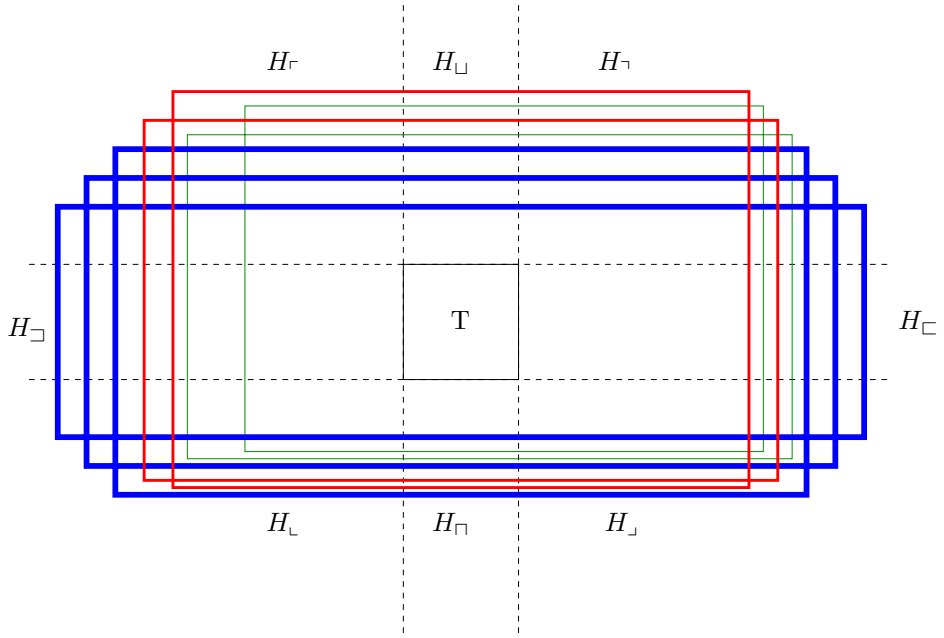


FIG. 7.13. *An illustration for the proof of Lemma* 7.12. *The thickest rectangles belong to all four chains. The second-thickest rectangles belong to the top-left and top-right chains, and the thinnest rectangles belong only to the top-right chain.*

In this section we show that for every nonorphan tile T, (i) $|sel(T)| \leq O(m)$, and (ii) $sel(T)$ can be CF-colored using $O(\log |sel(T)|)$ colors.

We begin by counting the number of rectangles in $sel(T)$. Since the side-length of every rectangle is in the range $[1, 2]$, and all the rectangles in $sel(T)$ are centered in T, it follows that $\bigcup_{S \in sel(T)} S$ intersects at most 25 tiles. Therefore, $|sel(T)| = O(m)$ for every tile T.

Similarly to Lemma 7.11, $|\chi_{\text{opt}}(\mathcal{R})| = \Omega(\log m)$. To obtain the constant-ratio approximation algorithm, we next show that $sel(T)$ can be CF-colored using $O(\log |sel(T)|)$ colors. Note that Theorem 1.5 is not applicable in this case since the rectangles are not congruent.

LEMMA 7.12. *Let $\mathcal{R}'$ be a set of axis-parallel rectangles with minimum width (height) at least* 1. *Assume that all centers of rectangles in $\mathcal{R}'$ reside in a square tile of side-length* 1/2. *Then it is possible to CF-color $\mathcal{R}'$ using $O(\log(|\mathcal{R}'|))$ colors.*

Proof. Let T be the $1/2 \times 1/2$ tile that contains the centers of the rectangles in $\mathcal{R}'$. Extend the sides of T into lines, and consider the subdivision of the plane into nine regions by these four lines. The subdivision consists of (i) the tile itself T, (ii) four corner regions denoted by $H_{\llcorner}$, $H_{\lrcorner}$, $H_{\urcorner}$, and $H_{\ulcorner}$, and (iii) four remaining regions denoted by $H_{\sqcup}$, $H_{\sqcap}$, $H_{\sqsubset}$, and $H_{\sqsupset}$. These regions are depicted in Figure 7.13.

Since each of the four regions $H_{\sqcup}$, $H_{\sqcap}$, $H_{\sqsubset}$, and $H_{\sqsupset}$ is of height/width 1/2, it suffices to select one rectangle for each and give it a unique color in order to serve the intersection of $\mathcal{R}'$ with each of them. In particular, for $H_{\sqcup}$ we take the rectangle whose top edge has the largest y coordinate; for $H_{\sqcap}$, the rectangle whose bottom edge has the smallest y coordinate, and similarly for $H_{\sqsubset}$ and $H_{\sqsupset}$. Any one of these (at most) four rectangles can serve all of T as well.

Next we observe that in order to serve each of the four corner regions $H_{\llcorner}$, $H_{\lrcorner}$, $H_{\ulcorner}$, and $H_{\urcorner}$, it suffices to focus on four corner-chains. Let $\widetilde{\mathcal{R}}'_{\llcorner}$, $\widetilde{\mathcal{R}}'_{\lrcorner}$, $\widetilde{\mathcal{R}}'_{\ulcorner}$, and $\widetilde{\mathcal{R}}'_{\urcorner}$, respectively, denote the set of rectangles that appear in the *envelope* of $\mathcal{R}'$ in each of the four corner parts. That is, those rectangles in $\mathcal{R}'$ whose corresponding corners ($\llcorner$ in $H_{\llcorner}$, $\lrcorner$ in $H_{\lrcorner}$, and, in general, γ in H_γ) are not contained in any other rectangle in $\mathcal{R}'$. The intersection of any other rectangle in $\mathcal{R}'$ with each H_γ, $\gamma \in \Gamma$, is contained in the intersection of the corresponding subset $\widetilde{\mathcal{R}}'_\gamma$ with H_γ. Note that the four subsets are not necessarily disjoint.

By a slight variant of Claim 7.3, each corner-chain $\widetilde{\mathcal{R}}'_{\mathrm{op}(\gamma)}$ is indeed a chain with respect to H_γ. While it is possible to apply Lemma 6.4 to each of these chains, we do not directly obtain a single consistent coloring because the different chains are not necessarily disjoint. Instead, we partition the rectangles into $2^4 - 1 = 15$ disjoint subsets, where each subset consists of rectangles that belong to the same nonempty subset of corner-chains (e.g., $\widetilde{\mathcal{R}}'_{\urcorner}$ and $\widetilde{\mathcal{R}}'_{\lrcorner}$ but not $\widetilde{\mathcal{R}}'_{\ulcorner}$ and $\widetilde{\mathcal{R}}'_{\llcorner}$).

The important observation regarding the envelope of $\mathcal{R}'$ is that if every boundary segment is given a symbol that corresponds to the rectangle it belongs to, then the sequence of symbols is a Davenport–Schinzel sequence $DS(n, 2)$ [SA95]. Namely, no two consecutive symbols are equal, and there is no alternating subsequence of length 4 (i.e., no "$\ldots a \ldots b \ldots a \ldots b \ldots$" for every pair of symbols $a \neq b$).

As a consequence, if two rectangles belong to more than one chain (that is to two, three, or even all four chains), then they appear *in the same order* (up to reversal) in all chains they belong to. Hence we can color each of the 15 subsets separately in a consistent manner (using 15 different palettes). The total number of colors used is hence $O(\log(|\widetilde{\mathcal{R}}|))$, as required. □

8. Coloring arrangements of regular hexagons. In this section we prove Theorem 1.5 for the case of regular hexagons. The proof follows the ideas used in the proof for the case of rectangles. We therefore provide a sketch of the proof (with accompanying illustrations) but do not give the full details of the proof.

8.1. Preliminaries. The sets of regular hexagons that we consider are axis-parallel; namely, two of the sides of the hexagons are parallel to the x-axis. The *type* of a vertex is determined by the slope of the segment connecting the center of the hexagon with the vertex (see Figure 8.1). In the same fashion, we define the type of an edge of the hexagon.

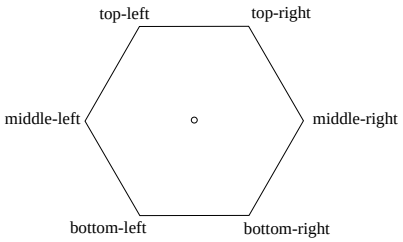


FIG. 8.1. *A hexagon and its vertices.*

The tiling. In the case of hexagons we consider a tiling of the plane by equilateral triangles with unit side-lengths; one side of each triangular tile is horizontal (see Figure 8.2). Triangular tiles have two possible orientations: In the *up* orientation, the vertex opposite the horizontal edge is above that edge, and in the *down* orientation, that vertex is below the horizontal edge.

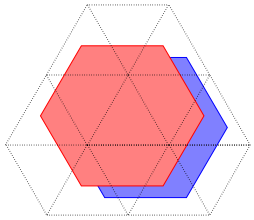


FIG. 8.2. *The triangular tiling and a pair of hexagons. The tile borders are depicted by dotted lines. Both centers of the hexagons are in the middle tile. Both hexagons completely cover the middle tile. Larger hexagons may intersect more than the* 12 *neighboring tiles.*

We adapt the notation of section 7 as follows. The set of hexagons is denoted by $\mathcal{H}$. We assume that the side-length of every hexagon is in the range $[1, \rho]$. For a tile T, we let $\mathcal{H}(T)$ denote the set of hexagons in $\mathcal{H}$ that belong to T (that is, whose center resides in T). A tile T is an *orphan* if $\mathcal{H}(T) = \emptyset$, and it is *bare* if no hexagon in $\mathcal{H}$ intersects it.

Since the tiles are equilateral triangles of side-length 1, the following holds for any set of hexagons $\mathcal{H}$ with side-lengths at least 1.

OBSERVATION 1. *For every tile T and hexagon $H \in \mathcal{H}$,* (i) *if $H \in \mathcal{H}(T)$, then $T \subset H$;* (ii) *T contains at most one vertex of H;* (iii) *if T intersects two edges e_1, e_2 of a hexagon H, then these edges are adjacent and T contains also the vertex $e_1 \cap e_2$.*

Disjoint palettes. As in the case of disks (cf. Theorem 1.2), we reduce the problem to the case of size-ratio 2 by paying a factor of $\log \rho$. Henceforth, we assume that $\rho \leq 2$. We assign a palette to every tile T. The colors assigned to hexagons in $\mathcal{H}(T)$ belong to the palette assigned to T. Palettes are disjoint, and the distribution of palettes is such that intersecting hexagons from different tiles are assigned different colors. This requires only a constant number of palettes. For example, consider a tiling of the plane with hexagonal supertiles that contain a constant number of triangular tiles. Assign every triangular tile within a hexagonal supertile a different palette, and extend this coloring periodically according to the hexagonal supertiles. The resulting assignment of palettes is as required.

8.2. Coloring arrangements of hexagons. As in the case of rectangles, the algorithm has two stages. In the first stage, each nonbare orphan tile T selects a subset of hexagons whose union serves the covered regions in T. For each nonorphan tile T, let $sel(T)$ denote the subset of hexagons in $\mathcal{H}(T)$ that were selected by orphan tiles in the first stage. In the second stage, the hexagons in $sel(T)$ are CF-colored, for every nonorphan tile T, using colors from the palette assigned to T.

8.2.1. Selection of hexagons by nonbare orphan tiles. Consider a nonbare orphan tile T. If there exists a hexagon that covers all of T, then we simply select one of these hexagons to serve it and no more selections are required. We now consider nonbare orphan tiles that are not covered by a single hexagon.

For an edge type e, let $\mathcal{H}(T, e)$ denote the set of hexagons that intersect T with an edge that is of type e (i.e., a nonempty intersection, but no vertex of the hexagon is contained in T). We claim that a single hexagon covers the intersection of T with hexagons from $\mathcal{H}(T, e)$. For example, let e be the top horizontal edge. The set of hexagons that intersect T with their top horizontal edge is hence denoted by $\mathcal{H}(T, e)$. Among these hexagons, pick the hexagon H with the highest center. The hexagon H covers the intersection of T with every hexagon in $\mathcal{H}(T, e)$. This completes

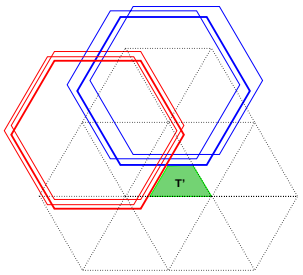


FIG. 8.3. *Let T be the central triangular tile in the figure, which is an orphan tile. The figure illustrates the choice of hexagons that intersect T with an edge. The selected hexagons are the two thick hexagons, and the region $T' \subset T$ that remains after their selection is filled.*

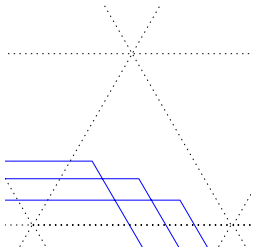


FIG. 8.4. *An example of a top-right chain. For simplicity, in this figure $T' = T$. That is, the dotted triangle is a tile T.*

the discussion of the selection of hexagons that intersect T with an edge. For an illustration, see Figure 8.3.

We denote by T' the region contained in T that remains after this choice of at most six hexagons (one per edge type). Note that if T' is nonempty, then T' is a polygon with at least three edges and at most six edges. The edges of Q are parallel to those of the hexagons in $\mathcal{H}$.

We now consider the selection of hexagons that intersect T' with a vertex. Let γ denote a vertex type (e.g., top-right), and let $\mathcal{H}(T', \gamma)$ denote the set of hexagons whose γ-vertex is in T'. Among the hexagons in $\mathcal{H}(T', \gamma)$, let $\widetilde{\mathcal{H}}(T', \gamma)$ denote the hexagons that participate in the envelope of $\mathcal{H}(T', \gamma)$ in T'. Similarly to the analysis in the case of rectangles, the latter cover all of the intersection of T' with the former, and furthermore they constitute a (corner) chain with respect to T'. We refer to the chain in terms of the vertex type (e.g., top-right chain). For an illustration, see Figure 8.4.

Thus, there are at most six corner-chains intersecting T', one for each vertex type. Here we have three types of "interactions" between chains, depending on the distance between the corresponding vertex types on the hexagons—that is, distance-one (e.g., top-right and top-left), distance-two (e.g., top-right and middle-left), and distance-three (e.g., top-right and bottom-left).

Interactions between distance-one and distance-two chains. Interactions between distance-one chains and distance-two chains are analogous to the interactions between corner-chains of adjacent corners in the case of rectangles. Specifically, for each such pair of chains, we can select a single hexagon from each chain so that (1) the union of the two selected hexagons covers the intersection between the chains, and (2) the remaining hexagons (not covered by the two selected hexagons) constitute disjoint

chains with respect to T' minus the two hexagons. For an illustration, see Figure 8.5.

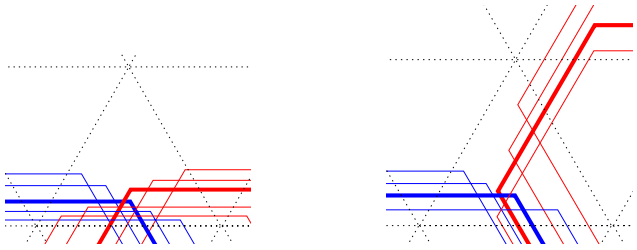


FIG. 8.5. *The two "benign" interactions between corner-chains in a tile T. On the left side is a distance-one interaction between a top-right chain and a top-left chain. On the right is a distance-two interaction between a top-right chain and a middle-left chain. In each figure, the two bold hexagons are those selected from the two chains.*

It follows that, by selecting at most four hexagons from each of the six chains that intersect T', it is possible to service all areas of intersections between such pairs of subsets of hexagons. Let $T'' \subseteq T'$ denote the remaining region in $T' \subseteq T$ that is not covered by these selected hexagons.

Interactions between pairs of distance-three (opposite) chains. The case of interactions between distance-three chains is analogous to the interaction between opposite chains in the case of rectangles. In particular, it is possible to select an approximately minimum subset of hexagons from the two chains so as to serve all the area in their union (within the region T''). For an illustration, see Figure 8.6.

8.3. Coloring the selected hexagons. We now return to each nonorphan tile T and assign colors to the hexagons requested from it. Note that Theorem 1.5 is not applicable since the hexagons are not congruent.

LEMMA 8.1. *Let $\widetilde{\mathcal{H}}$ be a subset of axis-aligned hexagons with side-lengths at least 1, which all belong to the same tile T. Then it is possible to CF-color $\widetilde{\mathcal{H}}$ using $O(\log(|\widetilde{\mathcal{H}}|)$ colors.*

Proof sketch. First, we assume, without loss of generality, that every hexagon in $\widetilde{\mathcal{H}}$ participated in the envelope (i.e., contains a vertex in $\bigcup_{H\in\widetilde{\mathcal{H}}} H$).

Similarly to the proof of Theorem 1.4 for rectangles, we extend the sides of a tile to partition the area covered by $\widetilde{\mathcal{H}}$ into several subregions (see Figure 8.7). The number of resulting regions is seven (including the tile T itself, which is covered by every hexagon in $\widetilde{\mathcal{H}}$). Three of these subregions have a common vertex with T (and are referred to as the "angular" subregions), and three have a common edge (and are referred to as the "trapeze" subregions). The vertices of every hexagon in $\widetilde{\mathcal{H}}$ are in the trapeze subregions. Hence, it is possible to select at most three hexagons to serve the angular subregions. Each of these hexagons is assigned a unique color (and thus T itself is also served).

We now deal with serving points in the trapeze regions. We wish to identify two chains in each trapeze region. Fix a trapeze region R. Every hexagon has two adjacent vertices in the trapeze region (as well as the edge connecting these vertices). Let u and v denote the vertex types that appear in R. Pick the hexagon H_R whose edge is farthest away from the corresponding edge of the triangular tile. Consider the sequence of vertices along the envelope of $\widetilde{\mathcal{H}}$ in R. This sequence starts with a block of vertices of type u and ends with a block of vertices of type v. The two vertices of H_R in R appear consecutively in this envelope. By picking H_R and assigning it a unique color, the envelope in R is separated into two parts. Moreover, the region

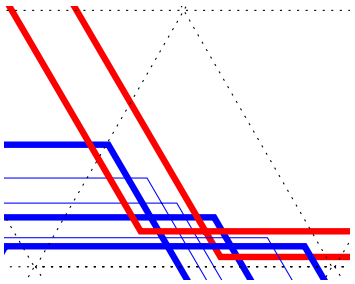


FIG. 8.6. *An interaction between the distance-three (opposite) chains top-right and bottom-left. The selected hexagons are bold.*

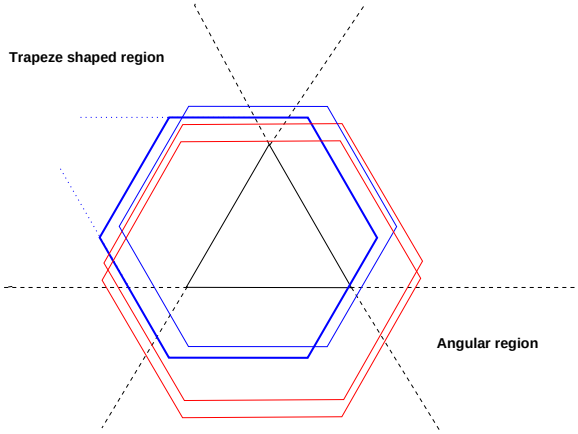


FIG. 8.7. *The partitioning of the area covered by hexagons that belong to the same tile. The six subregions H_R outside the tile are determined by the dashed lines that are extensions of sides of the tile. There are three angular regions, which can each be served by a single hexagon, and three trapeze-shaped regions. The two dotted lines within the top-left trapeze-shaped region are determined by the hexagon selected as in the proof (sketch) of Lemma* 8.1. *Each of the two dotted lines, paired with one of the dashed lines bounding the trapeze-shaped region, define the (angular) subregion that contains a (disjoint) corner-chain.*

$(R \setminus H_R) \cap (\bigcup_{H \in \tilde{\mathcal{H}}} H)$ consists of two disjoint connected parts. The hexagons whose vertices appear in the envelope in each part are chains with respect to $R \setminus H_R$. Thus, by picking at most six hexagons and assigning them unique colors, we have identified six disjoint chains.

As in the proof of Lemma 7.12, hexagons that belong to multiple chains appear in the same order (up to reversal) in these chains. Hence we partition the hexagons that appear in chains into at most $2^6 - 1$ subsets, where within each subset all hexagons belong to the same chains. (A finer counting argument is based on showing that for every three or more chains there can be at most one hexagon that belongs to all these chains. Hence we actually focus on subsets of hexagons that belong to one or two chains.) Each such subset is provided with a disjoint palette and can be colored using a logarithmic (in its size) number of colors. □

Finally, the proof of Theorem 1.4 for regular hexagons follows by combining the above lemma with a lower bound analogous to Lemma 7.10, the basic properties of the tiling, and the requesting process from orphan tiles.

9. Consequences.

9.1. Universal bounds for noncongruent rectangles and hexagons. As a corollary of Lemma 7.12 we also obtain a universal bound for the case of rectangles that is analogous to the case of disks (part 1 of Theorem 1.2).

Specifically, for each pair of integers $i, j \geq 1$, let $\mathcal{R}^{i,j}$ denote the subset of rectangles in $\mathcal{R}$ whose width is in the range $[2^{i-1}, 2^i)$ and whose height is in the range $[2^{j-1}, 2^j)$. Let $\phi_{2^i,2^j}(\mathcal{S}^{i,j})$ denote the maximum number of centers of rectangles in $\mathcal{R}^{i,j}$ that are contained in a rectangular tile of width 2^i and height 2^j. We refer to $\phi_{2^i,2^j}(\mathcal{R}^{i,j})$ as the *local density* of $\mathcal{R}^{i,j}$ (with respect to rectangular tiles of width 2^i and height 2^j).

THEOREM 9.1. *There exists an algorithm that, given a set $\mathcal{R}$ of axis-parallel rectangles with side-lengths in the interval $[1, \rho]$, finds a CF-coloring χ of $\mathcal{R}$ using $O\left(\min\left\{\sum_{i=1}^{\log(\rho)+1}\sum_{j=1}^{\log(\rho)+1}(1+\log\phi_{2^i,2^j}(\mathcal{R}^{i,j})), \log|\mathcal{R}|\right\}\right)$ colors.*

Lemma 8.1 implies an analogous theorem for hexagons.

9.2. CF-multicoloring. An interesting by-product of Theorem 1.4 and its analysis has to do with minimum *CF-multicoloring*. A CF-multicoloring of a collection $\mathcal{S}$ is a mapping χ from $\mathcal{S}$ to *subsets* of colors. The requirement is that for every point $x \in \bigcup_{S\in\mathcal{S}} S$ there exist a color i such that $\{S : x \in S,\ i \in \chi(S)\}$ contains a single subset. It has been observed by Bar-Yehuda ([B01], based on [BGI92]) that every set-system $(X, \mathcal{S})$ can be CF-multicolored using $O(\log|X| \cdot \log|\mathcal{S}|)$ colors. Since the problem of minimum graph coloring can be reduced to CF-coloring of set-systems, it follows that there exist set-systems for which there is an exponential gap between the minimum number of colors required in a CF-coloring and the minimum number of colors required in a CF-multicoloring. In particular, this is true when the set-system $(X, \mathcal{S})$ corresponds to a clique $G = (V, E)$ as follows: There is a set S_v for every vertex $v \in V$, and there is a point $x_e \in X$ for every edge $e \in E$. The set S_v contains the point x_e if and only if v is an endpoint of e. The number of colors required to CF-color this set-system is $|\mathcal{S}| = |V|$, in contrast to the $O(\log^2|\mathcal{S}|)$ colors that are sufficient for CF-multicoloring.

A natural question is whether, in the geometric setting that we study, the number of colors required for CF-multicoloring is significantly smaller than that required for CF-coloring. An example in which CF-multicoloring saves colors is a "circle" of five congruent squares such that every adjacent pair of squares intersects and no three squares intersect. Since the number of squares is odd, three colors are needed for CF-coloring. However, CF-multicoloring requires only two colors: Color the first square with two colors, and then color the rest of the squares with alternating colors. The lower bound proved in Lemma 6.3 also applies to CF-multicoloring, and hence CF-multicoloring does not save colors in chains. Furthermore, it follows from our analysis (cf. Lemma 7.10) that CF-multicoloring reduces the number of colors by at most a constant in the case of congruent squares (or hexagons).

THEOREM 9.2. *Let $\mathcal{S}$ denote a set of congruent axis-parallel squares, and let $\chi^{\text{multi}}_{\text{opt}}(\mathcal{S})$ denote an optimal CF-multicoloring of $\mathcal{S}$. Then $|\chi^{\text{multi}}_{\text{opt}}(\mathcal{S})| = \Theta(|\chi_{\text{opt}}(\mathcal{S})|)$.*

Acknowledgments. We thank the reviewers of this manuscript for careful reading and many helpful remarks. We thank an anonymous FOCS reviewer for suggesting that the NP-completeness of coloring intersection graphs of unit disks could be used to prove the NP-completeness of CF-coloring arrangements of unit disks. We thank Micha Sharir for helpful discussions.

REFERENCES

[AHK+01] K. Aardal, S. van Hoesel, A. Koster, C. Mannino, and A. Sassano, *Models and Solutions Techniques for Frequency Assignment Problem*, ZIB Report 01-40, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany, 2001; available online at http://www.zib.de/PaperWeb/abstracts/ZR-01-40/.

[AKM+01] Z. Abrams, J. Könemann, A. Meyerson, K. Munagala, and S. Plotkin, *Facility Location with Interference*, Working paper 2001-E23, GSTA, Carnegie Mellon University, Pittsburgh, PA, 2001.

[AH77a] K. Appel and W. Haken, *Every planar map is 4-colorable—1: Discharging*, Illinois J. Math., 21 (1977), pp. 421–490.

[AH77b] K. Appel and W. Haken, *Every planar map is 4-colorable—2: Reducibility*, Illinois J. Math., 21 (1977), pp. 491–567.

[BGI92] R. Bar-Yehuda, O. Goldreich, and A. Itai, *On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization*, J. Comput. System Sci., 45 (1992), pp. 104–126.

[B01] R. Bar-Yehuda, *personal communication*, Technion, Israel Institute of Technology, Haifa, Israel, 2001.

[BKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwartzkopf, *Computational Geometry—Algorithms and Applications*, Springer-Verlag, Berlin, New York, 1997.

[CCJ90] B. N. Clark, C. J. Colburn, and D. S. Johnson, *Unit disks graphs*, Discrete Math., 86 (1990), pp. 165–177.

[C69] H. S. M. Coxeter, *Introduction to Geometry*, 2nd ed., Wiley, New York, 1969.

[DBJC98] N. W. Dunkin, J. E. Bater, P. G. Jeavons, and D. A. Cohen, *Towards High Order Constraint Representations for the Frequency Assignment Problem*, Technical report CSD-TR-98-05, Computer Science Department, Royal Holloway, University of London, London, 1998; available online at http://www.dcs.rhbnc.ac.uk/research/constraints/publications/index.shtml.

[FK98] U. Feige and J. Kilian, *Zero knowledge and the chromatic number*, J. Comput. System Sci., 57 (1998), pp. 187–199.

[GGRV00] M. Galota, C. Glasser, S. Reith, and H. Vollmer, *A Polynomial-Time Approximation Scheme for Base Station Positioning in UMTS Networks*, Technical report 264, Institut für Informatik, Universität Würzburg, Würzburg, Germany, 2000.

[HS03] S. Har-Peled and S. Smorodinsky, *On conflict-free coloring of points and simple regions in the plane*, in Proceedings of the 19th Annual ACM Symposium on Computing Geometry, San Diego, CA, 2003, pp. 114–123.

[H01] X. Huang, *Automatic Cell Planning for Mobile Network Design: Optimization Models and Algorithms*, Ph.D. dissertation, Universität Karlsruhe, Karlsruhe, Germany, 2001.

[KMR01] S. O. Krumke, M. V. Marathe, and S. S. Ravi, *Models and approximation algorithms for channel assignment in radio networks*, invited paper in Wireless Networks, 7 (2001), pp. 575–584.

[MBH+95] M. V. Marathe, H. Breu, H. B. Hunt, III, S. S. Ravi, and D. J. Rosenkrantz, *Simple heuristics for unit disk graphs*, Networks, 25 (1995), pp. 59–68.

[PT03] J. Pach and G. Tóth, *Conflict free colorings*, in The Goodman Pollack Festschrift, Algorithms Combin. 25, B. Aronov, S. Basu, J. Pach, and M. Sharir, eds., Springer-Verlag, Berlin, 2003, pp. 665–671.

[SA95] M. Sharir and P. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, Cambridge, UK, 1995.

[SM03] S. Smorodinsky, *Combinatorial Problems in Computational Geometry*, Ph.D dissertation, Tel-Aviv University, Tel-Aviv, Israel, 2003.

[RSST96] R. Robertson, D. P. Sanders, P. Seymour, and R. Thomas, *Efficiently four-coloring planar graphs*, in Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, Philadelphia, PA, 1996, pp. 571–575.

SIAM J. COMPUT.
Vol. 33, No. 1, pp. 137–166

COMMUNICATION COMPLEXITY OF SIMULTANEOUS MESSAGES*

LÁSZLÓ BABAI†, ANNA GÁL‡, PETER G. KIMMEL§, AND SATYANARAYANA V. LOKAM¶

Abstract. In the multiparty communication game (CFL game) of Chandra, Furst, and Lipton [*Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, Boston, MA, 1983, pp. 94–99] k players collaboratively evaluate a function $f(x_0, \ldots, x_{k-1})$ in which player i knows all inputs except x_i. The players have unlimited computational power. The objective is to minimize communication.

In this paper, we study the SIMULTANEOUS MESSAGES (SM) model of multiparty communication complexity. The SM model is a restricted version of the CFL game in which the players are not allowed to communicate with each other. Instead, each of the k players simultaneously sends a message to a *referee*, who sees none of the inputs. The referee then announces the function value.

We prove lower and upper bounds on the SM complexity of several classes of explicit functions. Our lower bounds extend to randomized SM complexity via an entropy argument. A lemma establishing a tradeoff between average Hamming distance and range size for transformations of the Boolean cube might be of independent interest.

Our lower bounds on SM complexity imply an exponential gap between the SM model and the CFL model for up to $(\log n)^{1-\epsilon}$ players for any $\epsilon > 0$. This separation is obtained by comparing the respective complexities of the Generalized Addressing Function, $\mathsf{GAF}_{G,k}$, where G is a group of order n. We also combine our lower bounds on SM complexity with the ideas of Håstad and Goldmann [*Comput. Complexity*, 1 (1991), pp. 113–129] to derive superpolynomial lower bounds for certain depth-2 circuits computing a function related to the GAF function.

We prove some counterintuitive upper bounds on SM complexity. We show that $\mathsf{GAF}_{\mathbb{Z}_2^t,3}$ has SM complexity $O(n^{0.92})$. When the number of players is at least $c \log n$, for some constant $c > 0$, our SM protocol for $\mathsf{GAF}_{\mathbb{Z}_2^t,k}$ has polylog(n) complexity. We also examine a class of functions defined by certain depth-2 circuits. This class includes the Generalized Inner Product function and Majority of Majorities. When the number of players is at least $2 + \log n$, we obtain polylog(n) upper bounds for this class of functions.

Key words. communication complexity, circuit complexity, lower bounds, group theory

AMS subject classifications. 68Q05, 68Q17, 68R05

DOI. 10.1137/S0097539700375944

1. Introduction.

1.1. The model. Chandra, Furst, and Lipton [CFL83] introduced the following multiparty communication game: Let $f(x_0, \ldots, x_{k-1})$ be a Boolean function, where

*Received by the editors July 31, 2000; accepted for publication (in revised form) February 17, 2003; published electronically November 14, 2003. This is a significantly expanded version of the conference paper [BaKL95]. A substantial part of the present work was done while the last three authors were students at the University of Chicago.

http://www.siam.org/journals/sicomp/33-1/37594.html

†Department of Computer Science, University of Chicago, Chicago, IL 60637-1588 (laci@cs.uchicago.edu). This author's research was partially supported by NSF grants CCR-9014562 and CCR-9732205.

‡Department of Computer Science, University of Texas at Austin, Austin, TX 78712-1188 (panni@cs.utexas.edu). This author's research was partially supported by NSF CAREER Award CCR-9874862 and an Alfred P. Sloan Research Fellowship.

§Department of Computer Science, Northeastern Illinois University, Chicago, IL 60625-4699 (pgkimmel@neiu.edu).

¶EECS Department, University of Michigan, Ann Arbor, MI 48109-2122 (satyalv@eecs.umich.edu). This author's research was partially supported by NSF grant CCR-9988359. Part of this author's work was done while at the School of Mathematics, Institute for Advanced Study, Princeton, and was supported by NSF grant DMS 97-29992.

each x_i is a bit string of fixed length $\leq n$ bits. k players collaborate to evaluate $f(x_0, \ldots, x_{k-1})$. Each player has full knowledge of the function f. The ith player knows each input argument except x_i; we will refer to x_i as the input *missed* by player i. We can imagine input x_i written on the forehead of player i. Each player has unlimited computational power. They share a blackboard, viewed by all players, where in each round of the game some player writes a bit. The last bit written on the board must be the function value.

DEFINITION 1.1. *A* multiparty protocol *is a specification of which player writes in each round and what that player writes. The protocol must specify the following information for each possible sequence of bits that is written on the board so far:*

1. *Whether or not the game is over, and in case it is not over, which player writes the next bit: This information should be completely determined by the information written on the board so far.*
2. *What that player writes: This should be a function of the information written on the board so far and of the input seen by that player.*

The cost *of a multiparty protocol is the number of bits written on the board for the worst case input. The* multiparty communication complexity *of f, denoted $C(f)$, is the minimum cost of a protocol computing f.*

Fairly strong multiparty communication complexity lower bounds of the form n/c^k were obtained by Babai, Nisan, and Szegedy [BaNS92] for some families of explicit functions. However, it seems that those methods do not extend to logarithmic number of players and beyond.

Håstad and Goldmann [HG91] found a curious application of the [BaNS92] bounds to lower bounds for small depth threshold circuits. Subsequent work by Yao [Ya90] and Beigel and Tarui [BeT91] reduces ACC circuits (bounded depth, polynomial size circuits with Boolean and $\mathsf{MOD}\ m$ gates) to small depth circuits similar to those considered by [HG91]. These results imply that a superpolylogarithmic lower bound for the communication complexity of a function f with a superpolylogarithmic number of players would show that $f \notin \mathrm{ACC}$.

In fact, this separation would already follow from similar lower bounds in a weaker model, which we call the SIMULTANEOUS MESSAGES (SM) model (see Definition 2.1). This connection was pointed out to us by Avi Wigderson.

The SM model is a restricted version of the general multiparty communication model in which the players are not allowed to communicate with each other. Instead, each player can send a single message to a *referee*, who sees none of the inputs. The referee announces the value of the function based on the messages sent by the players.

The subject of this paper is the complexity of explicit functions in the SM model.

1.2. Lower bounds. We prove lower bounds on the SM complexity of the Generalized Addressing Function, $\mathsf{GAF}_{G,k}$, where G is a group of order n (see Definition 2.3). The input to $\mathsf{GAF}_{G,k}$ consists of $n + (k-1)\log n$ bits partitioned among the players as follows: player 0 gets a function $x_0 : G \longrightarrow \{0,1\}$ (represented as an n-bit string) on her forehead, whereas players 1 through $k-1$ get group elements $x_1, \ldots, x_{k-1}$, respectively, on their foreheads. The output of $\mathsf{GAF}_{G,k}$ for this input is the value of the function x_0 on the product $x_1 \cdot \ldots \cdot x_{k-1}$.

Our first result is an $\Omega(\frac{|G|^{1/(k-1)}}{k-1})$ lower bound on the SM complexity of $\mathsf{GAF}_{G,k}$ for any finite group G (Theorem 2.8).

The result uses a decomposition theorem for finite groups (Theorem 2.19). A related body of work, going back to a 1937 conjecture of Rohrbach [Ro37a, Ro37b], is discussed in a separate section, partly for the sake of its own interest (section 7).

In section 3, we prove a lower bound similar to Theorem 2.8 on the *randomized* SM complexity of $\mathsf{GAF}_{G,k}$. Specifically, we show that any randomized SM protocol for $\mathsf{GAF}_{G,k}$ with a success probability $\geq (1+\epsilon)/2$ must have a cost of $\Omega(\frac{|G|^{1/(k-1)}\epsilon^2}{k-1})$. The proof of this result is based on an "entropy loss lemma," which may be of independent interest (Lemmas 3.7 and 3.9). The lemma provides a tradeoff between the *average* Hamming distance travelled and the number of destinations reached by a transformation of the Boolean cube. Similar entropy based tools have been used earlier in [Man98] and [KaT00].

It is easy to see that the general multiparty communication complexity of $\mathsf{GAF}_{G,k}$ is at most $\log n + 1$. Hence the lower bounds stated above show that the SM model is exponentially weaker than the general model for up to $(\log n)^{1-\epsilon}$ players. In fact, we prove this *exponential gap* between the SM model and an intermediate model called the "one-way communication model" [NW93] (see Definition 2.6). This result supports the hope that it might be easier to obtain stronger lower bounds in the SM model than in the general communication model. On the other hand, as mentioned in section 1.1, sufficiently strong lower bounds in the SM model still have some of the same interesting consequences to circuit complexity as those in the general model.

As mentioned before, Håstad and Goldmann [HG91] relate lower bounds on multiparty communication complexity to lower bounds on certain depth-2 circuits. In this paper, we use ideas from [HG91] to relate SM complexity to depth-2 circuit complexity. In particular, we show that a circuit with an arbitrary symmetric gate at the top and AND gates at the bottom computing an explicit function on n variables derived from $\mathsf{GAF}_{\mathbb{Z}_2^t,k}$ must have size $\exp((\log n/\log\log n)^2)$. We note that similar techniques applying the general multiparty communication complexity lower bounds were used by Razborov and Wigderson [RaW93].

1.3. Upper bounds. A curious development concerning SM complexity is the discovery of unexpected upper bounds. It appeared natural to expect that, when the number of players is constant, the SM complexity of GAF should be $\Omega(n)$. This, however, is false. In fact, we give a counterintuitive upper bound of $O(n^{0.92})$ on the SM complexity[1] of $\mathsf{GAF}_{\mathbb{Z}_2^t,3}$. More generally, we show an upper bound of roughly $n^{O(\log k/k)} + \log n$ on the SM complexity of $\mathsf{GAF}_{\mathbb{Z}_2^t,k}$. This gives a polylog$(n)$ upper bound when the number of players is $\log n$; in fact, if the number of players is greater than $\log n$, an upper bound of $2 + \log n$ holds (sections 5.1 and 5.2).

The $O(n^{0.92})$ upper bound, first published in [BaKL95] (a preliminary version of this paper), together with related results about cyclic groups $\mathbb{Z}_n$ by Pudlák and Rödl [PR93] and Pudlák, Rödl, and Sgall [PRS97], has prompted a refinement of an approach proposed by Nisan and Wigderson [NW93] toward superlinear size lower bounds on log-depth circuits. This approach uses 3-party communication complexity lower bounds and exploits a graph-theoretic reduction due to Valiant [Va77], building on earlier results by Erdős, Graham, and Szemerédi [EGS75]. To explain this approach, let us consider a Boolean function $f_1 : \{0,1\}^{O(n)} \times \{0,1\}^{O(n)} \times \{0,1\}^{\log n} \longrightarrow \{0,1\}$. From this, let us construct an n-output function $f(x,y) = (z_1,\ldots,z_n)$ by setting $z_j := f_1(x,y,j)$. Then an $\Omega(n)$ lower bound on the *total* number of bits communicated in a 3-party SM protocol for f_1 (with x, y, and j on the foreheads of players 0, 1, and 2, respectively) would imply a superlinear lower bound on log-depth circuits computing the function f. In particular, if there were an $\Omega(n)$ lower bound on the 3-party SM communication complexity of $\mathsf{GAF}_{\mathbb{Z}_2^t,3}$, where $n = 2^t$, then the above

[1] This bound has subsequently been improved to $O(n^{0.73})$ in [AmL00].

connection would have yielded an explicit function requiring superlinear size circuits of log-depth.

However, we have a 3-party SM protocol for $\mathsf{GAF}_{\mathbb{Z}_2^t,3}$ that uses only $n^{0.92}$ bits of communication. Analogously, [PR93, PRS97] prove an $o(n)$ upper bound on the 3-party SM complexity of $\mathsf{GAF}_{\mathbb{Z}_n,k}$. On the other hand, these and several similar functions are conjectured to require superlinear size circuits of log-depth. This situation motivated a refined version of the approach from [NW93]. This refined version seeks 3-party SM lower bounds when there are certain constraints on the lengths of messages from individual players. Indeed, in response to the surprising upper bounds from [BaKL95] and [PR93], Kushilevitz and Nisan present such an approach in their book [KuN97, section 11.3]. We describe this new formulation below.

Let f be an n-bit output function and f_1 its single-bit output counterpart as defined above. Then Valiant's lemma implies the following: if f has log-depth linear size circuits, then f_1 has an SM protocol in which, for any fixed $\epsilon > 0$, (i) player 2 sends at most $O(n/\log\log n)$ bits, and (ii) players 0 and 1 send $O(n^\epsilon)$ bits each. Thus a lower bound showing that any 3-party SM protocol for an explicit f_1 must violate (i) or (ii) would yield an explicit f that cannot be computed simultaneously in linear size and logarithmic depth.

It is interesting to note that, in contrast to the lower bound, our upper bound depends heavily on the structure of the elementary abelian 2-group $G = \mathbb{Z}_2^t$. In particular, the upper bound does not apply to the cyclic group $G = \mathbb{Z}_n$. For $\mathsf{GAF}_{\mathbb{Z}_n,k}$, Pudlák, Rödl, and Sgall [PRS97] prove an upper bound of $O(n\log\log n/\log n)$ for $k = 3$ (three players) and of $O(n^{6/7})$ for $k \geq c\log n$. Their upper bounds have been significantly improved by Ambainis [Am96] to $O(\frac{n\log^{1/4} n}{2^{\sqrt{\log n}}})$ for $k = 3$ and to $O(n^\epsilon)$ for an arbitrary $\epsilon > 0$ for $k = O((\log n)^{c(\epsilon)})$. However, these bounds for $\mathbb{Z}_n$ are still much weaker than the corresponding bounds for $\mathbb{Z}_2^t$ presented in this paper.

We also give surprising upper bounds on the SM complexity of a different class of functions defined by certain depth-2 circuits (section 6). For this class of functions, we prove polylog(n) upper bounds on the SM complexity when the number of players is at least $\log n + 2$. This class of functions includes Generalized Inner Product (GIP) and Majority of Majorities. The special case of GIP improves a result due to Grolmusz [G94], where the same upper bound is given for 2-round protocols. We note that GIP is a prime example in the study and applications of multiparty communication complexity [BaNS92, G94, HG91, RaW93]. Majority of Majorities is an interesting candidate function to be outside ACC.

The circuits defining the class of functions mentioned above have an arbitrary symmetric gate of fan-in n at the top and gates of fan-in k (= number of players) at the bottom. Furthermore, each bottom gate is assumed to compute a symmetric function with very small 2-party, one-way communication complexity (we call such functions *compressible*; see Definition 6.1). We partition the input so that each player misses one bit from each bottom gate.

We also give an example of an explicit symmetric function that is not compressible (in the sense of Definition 6.1). The proof of this result uses Weil's character sum estimates.

1.4. Comparison with [BaKL95]. Finally, a comment on how the present paper relates to its preliminary version [BaKL95]. Most results of [BaKL95] have been superseded both in generality and in the elegance of the proof. This is especially true for the (deterministic and randomized) lower bounds which have been extended to all groups. A discussion of circuit complexity applications has been added. The

main new additions are the counterintuitive upper bounds for the case of more than $\log n$ players for a significant class of functions, including the Majority of Majorities function.

1.5. Organization of the paper. In section 2, we introduce the model of SM and prove a lower bound on the SM complexity of the Generalized Addressing Function (GAF) with respect to an arbitrary finite group G (see Definition 2.3). Section 3 extends this lower bound to the randomized SM complexity of $\mathsf{GAF}_{G,k}$. In section 4, we present some consequences of our lower bounds on SM complexity to certain depth-2 circuits. Sections 5 and 6 deal with upper bounds on SM complexity. In section 5, we give nontrivial upper bounds for GAF with respect to elementary abelian 2-groups, whereas in section 6 we define a natural class of functions and show very efficient SM protocols for them. In section 7, we discuss a group-decomposition problem arising from the GAF lower bounds; this section may be of interest in its own right. Section 8 concludes the paper with several open problems.

2. A Simultaneous Messages lower bound. Let $f(x_0, \ldots, x_{k-1})$ be a Boolean function, where each x_i is a bit string of fixed length $\leq n$. A referee and k players collaborate to evaluate $f(x_0, \ldots, x_{k-1})$. Each participant (referee and players) has full knowledge of the function f. For $0 \leq i \leq k-1$, the ith player, p_i, knows each input argument except x_i. The referee does not know any of the input. Each player p_i simultaneously passes a message of fixed length to the referee, after which the referee announces the function value. Each participant is a function of the arguments it "knows."

Definition 2.1. *An* SM protocol P for f *is a set of players along with a referee that correctly computes f on all inputs. The* cost *of an SM protocol for f is the length of the longest message sent to the referee by any individual player.*[2] *The* SM complexity *of f, denoted $C_0(f)$, is the minimum cost of a protocol computing f.*

Remark 2.2. This model is implicit in the work of Nisan and Wigderson [NW93, Theorem 7], where they consider the case $k = 3$. The first papers investigating the SM model in detail are the conference version of the current paper [BaKL95] and a paper by Pudlák, Rödl, and Sgall [PRS97]; the latter uses the name "oblivious communication complexity."

2.1. The Generalized Addressing Function $\mathsf{GAF}_{G,k}$. The function that we use to show an exponential gap between the SM and general multiparty communication models is the GAF, defined as follows.

Definition 2.3. *Let G be a group of order n. Elements of G are represented by binary strings of length* $\log n$. *Let $x_0 : G \longrightarrow \{0,1\}$ be a function represented as an n-bit string, and let $x_1, \ldots, x_{k-1} \in G$. Then the* Generalized Addressing Function *for the group G and k players, denoted by $\mathsf{GAF}_{G,k}$, is defined as follows:*

$$\mathsf{GAF}_{G,k}(x_0, \ldots, x_{k-1}) := x_0[x_1 \cdot \ldots \cdot x_{k-1}].$$

Here $\cdot$ denotes the group operation in G.

The notation $C_0(\mathsf{GAF}_{G,k})$ refers to the SM complexity of the $\mathsf{GAF}_{G,k}$ function under the natural partition of the input among the players; i.e., player i misses input x_i. Note that the partition of the input among the players is not balanced since

[2] This definition of the cost, as the ℓ_∞-norm of the vector of message lengths of the players, differs from that of the STACS '95 version [BaKL95], where we consider the ℓ_1-norm. We continue to use the total communication for C and C_1 (Definitions 1.1 and 2.6).

player 0 has $|x_0| = n$ bits "on her forehead," whereas player i for $1 \le i \le k-1$ has $|x_i| = \log n$ bits on her forehead.

Recall that $C(f)$ denotes the k-party communication complexity of the function f (where f is a function in k variables) (see section 1.1).

Observation 2.4. $C(\mathsf{GAF}_{G,k}) \le \log n + 1$.

Proof. Player p_0 writes $g = x_1 \cdot \ldots \cdot x_{k-1}$; then p_1 writes $x_0[g]$. □

Remark 2.5. This is a special case of the observation that $C(f) \le 1 +$ the length of the shortest input.

DEFINITION 2.6. *A special case of the communication model is* one-way communication*, in which each player may write on the blackboard only once, and they proceed in the prescribed order $p_0, p_1, \ldots, p_{k-1}$. Let $C_1(f)$ denote the* one-way communication complexity of f.

Clearly, $n \ge C_0(f) \ge C_1(f)/k \ge C(f)/k$ for any function f of k variables. For $\mathsf{GAF}_{G,k}$, the proof of Observation 2.4 gives a one-way protocol, so we obtain the following consequence.

COROLLARY 2.7. $C_1(\mathsf{GAF}_{G,k}) \le \log n + 1$. □

The main result of this section is an SM lower bound on the Generalized Addressing Function of the form $\Omega(n^{1/(k-1)}/(k-1))$. This bound implies an exponential separation between $C_0(f)$ and $C_1(f)$ (and hence also between $C_0(f)$ and $C(f)$) for up to $k = (\log n)^{1-\epsilon}$ players. We state the result.

THEOREM 2.8. *For any group G of order n and any $k \ge 2$,*

$$C_0(\mathsf{GAF}_{G,k}) \ge \frac{cn^{1/(k-1)}}{k-1},$$

where $c = (1 - 1/\sqrt{\mathrm{e}})/2 > 0.19$.

The proof of this lower bound is given in the next two subsections.

Remark 2.9. For $k = 3$, Theorem 2.8 gives an $\Omega(\sqrt{n})$ lower bound. Nisan and Wigderson [NW93] give an $\Omega(\sqrt{n})$ lower bound for a different function, based on hash functions [MNT93]. They actually show this lower bound for one-way complexity, establishing an exponential gap between $C_1(f)$ and $C(f)$ for $k = 3$.

Remark 2.10. Theorem 2.8 states an SM lower bound for all finite groups G. Special cases of this result were found independently by Pudlák, Rödl, and Sgall (the cyclic group $G = \mathbb{Z}_n$) [PRS97, Proposition 2.3] and by the authors of the conference version of the present paper (the elementary abelian group $G = \mathbb{Z}_2^t$) [BaKL95]. Those bounds were extended in a preliminary version of this paper to a large class of groups, including all solvable groups. For arbitrary groups, however, our original lower bound was worse by a logarithmic factor than the bound stated in Theorem 2.8. We express our gratitude to an anonymous referee for pointing out that a simple modification of our argument yields the improved result stated as Theorem 2.8.

All proofs use essentially the same strategy, an information-theoretic argument combined with a group-decomposition result. Simple cases of the group-decomposition result are discussed as Examples 2.14 and 2.15. The general group-decomposition theorem appears as Theorem 2.19.

2.2. SM lower bound for $\mathsf{GAF}_{G,k}$ and group decompositions. In this subsection, we give an SM lower bound for $\mathsf{GAF}_{G,k}$ in terms of a parameter ρ of G and k related to optimal decompositions of a large fragment of a group. In the next subsection we shall estimate ρ within a constant factor. From this bound, our SM lower bound for $\mathsf{GAF}_{G,k}$ (Theorem 2.8) will be immediate.

DEFINITION 2.11. *For a finite group G and $A, B \subseteq G$, the product AB is defined as $AB = \{a \cdot b : a \in A,\ b \in B\}$. Note that $|AB| \leq |A| \cdot |B|$.*

DEFINITION 2.12. *Let α be a real number, $0 < \alpha \leq 1$. For a finite group G and a positive integer u we define*

$$\rho_\alpha(G, u) = \min_{H_1, \ldots, H_u \subseteq G} \{\rho : |H_1 \cdot \ldots \cdot H_u| \geq \alpha |G| \text{ and } \forall i, |\widehat{H}_i| \leq \rho\},$$

where $\widehat{H}_i$ is defined to be the Cartesian product of all H_j except H_i.

Remark 2.13. Note that $|\widehat{H}_i| = \prod_{j \neq i} |H_j|$. Also note that $\widehat{H}_i$ is not the product $\prod_{j \neq i} H_j$ in the sense of Definition 2.11; in fact, $\widehat{H}_i$ is not even a subset of G. (It is a subset of $G \times \cdots \times G$ ($u - 1$ times).)

The following two examples give upper bounds on ρ for two special groups. In section 7, we will see that these upper bounds are optimal to within a constant factor.

EXAMPLE 2.14. *$\rho_1(\mathbb{Z}_2^t, u) \leq 2n^{1-1/u}$, where $n = 2^t$.*

Proof. Let $V = \mathbb{Z}_2^t$. Decompose V into a direct sum of u subspaces: $V = H_1 \oplus \cdots \oplus H_u$, where for each i, $1 \leq i \leq u$, $\lfloor t/u \rfloor \leq \dim H_i \leq \lceil t/u \rceil$. This implies

$$\begin{aligned} \rho_1(V, u) &\leq \max_i |\widehat{H}_i| = \max_i \frac{n}{|H_i|} = \frac{n}{\min_i |H_i|} \\ &\leq \frac{n}{2^{\lfloor t/u \rfloor}} \leq \frac{2n}{2^{t/u}} = \frac{2n}{n^{1/u}} = 2n^{1-1/u}. \end{aligned} \tag{2.1}$$

□

EXAMPLE 2.15. *$\rho_{1/2}(\mathbb{Z}_n, u) \leq 2n^{1-1/u}$ and $\rho_1(\mathbb{Z}_n, u) \leq 4n^{1-1/u}$.*

Proof. Let $2^t < n \leq 2^{t+1}$. We consider the elements of $\mathbb{Z}_n$ to be binary strings of length $t + 1$. Let K be the subset of $\mathbb{Z}_n$ given by binary strings with their most significant ($(t+1)$st) bit equal to zero. Identify K with $H_1 \oplus \cdots \oplus H_u$, where H_i is the set of binary numbers with t digits in which all digits are 0 except for the ith segment of $\lfloor t/u \rfloor$ or $\lceil t/u \rceil$ digits, which can be either 0 or 1. Thus each H_i has size $\geq 2^{\lfloor t/u \rfloor}$. Clearly, $|K| = 2^t \geq n/2$. By (2.1), we have that $\max_i |\widehat{H}_i| \leq 2|K|^{1-1/u} \leq 2n^{1-1/u}$. Hence $\rho_{1/2}(\mathbb{Z}_n, u) \leq 2n^{1-1/u}$.

To cover the entire group $\mathbb{Z}_n$, apply the above argument to bit strings of length $(t + 1)$ (but perform group operations in $\mathbb{Z}_n$). Then we get that $\max_i |\widehat{H}_i| \leq 2 \cdot 2^{(t+1)(1-1/u)} \leq 4n^{1-1/u}$, and this gives us the bound on $\rho_1(\mathbb{Z}_n, u)$.

As a concrete example, consider $\mathbb{Z}_{25}$, $u = 3$, and $\alpha = 1$. It is easy to see that a (complete) cover is given by $\mathbb{Z}_{25} = \{0, 16\} + \{0, 4, 8, 12\} + \{0, 1, 2, 3\}$. Note that in this cover, some elements (e.g., 5) have more than one factorization (under the group operation addition modulo 25). □

LEMMA 2.16. *For any finite group G and any α $(0 < \alpha \leq 1)$,*

$$C_0(\mathsf{GAF}_{G,k}) \geq \frac{\alpha |G|}{(k-1)\, \rho_\alpha(G, k-1)}.$$

Proof. We prove a lower bound on the length of the longest message sent by $p_1, \ldots, p_{k-1}$. We ignore p_0 by assuming that the referee knows whatever p_0 knows. This assumption can only make our lower bound stronger.

The proof is by an information-theoretic argument. Pick a factorization $K = \prod_{i=1}^{k-1} H_i$ of some subset $K \subseteq G$ which is optimal in the sense that $|K| \geq \alpha |G|$, $H_1, \ldots, H_{k-1} \subseteq G$, and $\max_i |\widehat{H}_i| = \rho_\alpha(G, k-1)$.

We shall restrict player p_0's inputs to functions $x_0 : G \longrightarrow \{0, 1\}$ such that $x_0(g) = 0$ for all $g \in G \setminus K$. For $i = 1, \ldots, k-1$, we shall restrict player p_i's inputs

to H_i. For a fixed x_0 (on p_0's forehead and hence visible to all p_i, $1 \le i \le k-1$), player p_i can send at most $|\widehat{H}_i|$ different messages. Hence the total number of bits received by the referee (for all combinations of these restricted inputs) is at most $\rho_\alpha(G, k-1)(k-1)\ell$, where ℓ is the length of the longest message sent by the players p_i, $1 \le i \le (k-1)$. But this collection of messages must determine every bit of x_0 corresponding to the set $K \subseteq G$. Hence we must have $\rho_\alpha(G, k-1)(k-1)\ell \ge \alpha n$, giving the claimed bound. The foregoing ideas are formalized below.

Let P be any SM protocol for $\mathsf{GAF}_{G,k}$. Let r denote the referee function. Let ℓ be the cost of P. For notational convenience, we assume, without loss of generality, that each player sends a message of length ℓ (padding their messages if necessary).

We define a function F in terms of the player and referee functions. The input to F will be a binary string of length $(k-1)\rho_\alpha(G, k-1)\ell$, and the output will be a $|K|$-bit string. We will then show that F is surjective, which will yield the theorem.

DEFINITION 2.17. *For each $g \in K$, we fix elements $h_1 \in H_1, \ldots, h_{k-1} \in H_{k-1}$ such that $g = h_1 \cdot \ldots \cdot h_{k-1}$ and refer to this as "the"* decomposition *of g.*

Now we define a function $F : \{0,1\}^{\ell|\widehat{H}_1|} \times \cdots \times \{0,1\}^{\ell|\widehat{H}_{k-1}|} \longrightarrow \{0,1\}^K$ as follows. Let $(w_1, w_2, \ldots, w_{k-1})$ be an input to F, where $w_i \in \{0,1\}^{\ell|\widehat{H}_i|}$. The ℓ-bit substrings of w_i are indexed by elements of $\widehat{H}_i$. The output bit $y(g)$ corresponding to $g \in K$ is determined as follows. Let $g = h_1 \cdot \ldots \cdot h_{k-1}$ be the decomposition of g. For $1 \le i \le k-1$, let $m_i(g)$ be the ℓ-bit substring of w_i at index $\widehat{h}_i := (h_1, \ldots, h_{i-1}, h_{i+1}, \ldots, h_{k-1}) \in \widehat{H}_i$. Let $m_0(g) = p_0(h_1, \ldots, h_{k-1})$. Now define $y(g) = r(m_0(g), m_1(g), \ldots, m_{k-1}(g))$.

CLAIM 2.18. *F is surjective.*

Proof. Let $x_0 \in \{0,1\}^G$ such that $x_0(g) = 0$ for all $g \in G \setminus K$. Define the input $w_{x_0} = (w_1, \ldots, w_{k-1})$ to F as follows: For each i, $1 \le i \le k-1$, and each $\widehat{h}_i \in \widehat{H}_i$, define the ℓ-bit substring of w_i at index $\widehat{h}_i$ to be $p_i(x_0, \widehat{h}_i)$, i.e., the message sent by p_i when she sees x_0 and $\widehat{h}_i$. We now show that $F(w_{x_0}) = x_0^K$ (the restriction of x_0 to K), which will prove the claim. Recalling our notation that $y = F(w_{x_0})$, we want to show that for each $g \in K$, $y(g) = x_0(g)$.

Let $g \in K$, and let $g = h_1 \cdot \ldots \cdot h_{k-1}$, $h_i \in H_i$ be the decomposition of g.

From the definition of w_{x_0}, for $1 \le i \le k-1$, we have $m_i(g) = (\ell$-bit substring of w_i at index $\widehat{h}_i) = p_i(x_0, \widehat{h}_i)$, and also $m_0(g) = p_0(h_1, \ldots, h_{k-1})$. Now using the definition of F we have

$$\begin{aligned} y(g) &= r(m_0(g), m_1(g), \ldots, m_{k-1}(g)) \\ &= r(p_0(h_1, \ldots, h_{k-1}), p_1(x_0, h_2, \ldots, h_{k-1}), \ldots, p_{k-1}(x_0, h_1, \ldots, h_{k-2})) \\ &= \mathsf{GAF}_{G,k}(x_0, h_1, \ldots, h_{k-1}) \qquad \text{by the correctness of the protocol} \\ &= x_0(g) \qquad \text{by definition of } \mathsf{GAF}_{G,k}. \end{aligned}$$

Thus $F(w_{x_0}) = x_0^K$ and F is surjective, and this completes the proof of the claim. □

Claim 2.18 implies that the domain of F is at least as large as the range of F. Thus $\ell(k-1)\rho_\alpha(G, k-1) \ge \ell(|\widehat{H}_1| + \cdots + |\widehat{H}_{k-1}|) \ge |K| \ge \alpha|G|$, and hence $\ell \ge n/(k-1)\rho_\alpha(G, k-1)$. This completes the proof of Lemma 2.16. □

2.3. Decomposition of groups. In this subsection we estimate the quantity $\rho_\alpha(G, u)$ for a specific positive constant α.

THEOREM 2.19. *Given a finite group G and a positive integer u, there exist subsets $H_1, \ldots, H_u \subseteq G$ such that $|\widehat{H}_i| < 2|G|^{1-1/u}$ for $i = 1, \ldots, u$, and $|H_1 \cdot \ldots \cdot H_u| >$*

$(1-1/\sqrt{\mathrm{e}})|G| > 0.39|G|$.

COROLLARY 2.20. *Let $\alpha = 1-1/\sqrt{\mathrm{e}} \approx 0.39$. Then for any finite group G and any positive integer u,*

$$\rho_\alpha(G,u) < 2|G|^{1-1/u}.$$

Combining Corollary 2.20 and Lemma 2.16, our lower bound for the SM complexity of $\mathsf{GAF}_{G,k}$ (Theorem 2.8) is immediate. □

For the proof of Theorem 2.19, we use the following result. Let G be a group and $a_1,\dots,a_k \in G$. The *cube* based on the sequence $a_1,\dots,a_k$ is the set $C(a_1,\dots,a_k) := \{1,a_1\}\cdot\ldots\cdot\{1,a_k\}$. In other words, $C(a_1,\dots,a_k)$ consists of the 2^k *subproducts* $a_1^{\epsilon_1}\dots a_k^{\epsilon_k}$, where $\epsilon_i \in \{0,1\}$.

THEOREM 2.21 (see [BaE82]). *Let G be a finite group of order n, and let ℓ be a positive integer. Then there exists a sequence of elements $a_1,\dots,a_\ell \in G$ such that $|C(a_1,\dots,a_\ell)| > n(1-\exp(-2^\ell/n))$.*

For completeness we include the proof.

LEMMA 2.22. *Let A be a subset of a finite group G. Then for some $x \in G$ we have*

$$\frac{|G\setminus(A\cup Ax)|}{|G|} \le \left(\frac{|G\setminus A|}{|G|}\right)^2. \tag{2.2}$$

Proof. Let $|G|=n$ and $|A|=k$. Let us select $x\in G$ at random from the uniform distribution. Then for every $g\in G$, the probability that $g\notin Ax$ is $(n-k)/n$. Therefore the expected number of those $g\in G\setminus A$ which do not belong to Ax is $(n-k)^2/n$. So this is the expected size of the set $G\setminus(A\cup Ax)$. Pick an x for which $|G\setminus(A\cup Ax)|$ is not less than its expected value. □

Proof of Theorem 2.21. We choose $a_1,\dots,a_\ell \in G$ successively as follows. Set $A_1=\{a_1\}$ and $A_{i+1}=A_i\cup A_i a_{i+1}$. Let $a_1\in G$ be arbitrary; given $a_1,\dots,a_i$, we choose $a_{i+1}\in G$ so as to maximize $|A_{i+1}|$.

Let $p_i = |G\setminus A_i|/n$.

We have $p_1 = 1-1/n$, and by Lemma 2.22 we have $p_{i+1}\le p_i^2$. Therefore

$$p_\ell \le \left(1-\frac{1}{n}\right)^{2^\ell} < \exp\left(-2^\ell/n\right). \tag{2.3}$$

Noting that $|C(a_1,\dots,a_\ell)| = n(1-p_\ell)$ completes the proof. □

Proof of Theorem 2.19. Let $n=|G|$, and let ℓ denote the integer satisfying $n/2 \le 2^\ell \le n$. Let $a_1,\dots,a_\ell\in G$ be the sequence of ℓ elements in G guaranteed by Theorem 2.21.

Let us split ℓ into u parts as evenly as possible: $\ell = k_1+\dots+k_u$, where $k_i \in \{\lfloor \ell/u\rfloor, \lceil \ell/u\rceil\}$. Let $H_j = C(a_{k_1+\dots+k_{j-1}+1},\dots,a_{k_1+\dots+k_j})$. (So H_1 is the cube based on the first k_1 members of the sequence $\{a_i\}$; H_2 is the cube based on the next k_2 members of the sequence, etc.) Then $H_1\dots H_u = C(a_1,\dots,a_\ell)$, and therefore $|H_1\dots H_u| > n(1-\exp(-2^\ell/n)) \ge n(1-1/\sqrt{\mathrm{e}})$.

Moreover, $|\widehat{H_i}| = 2^{\ell-k_i} < 2^{\ell(1-1/u)+1} \le 2n^{1-1/u}$. □

3. Randomized complexity of SIMULTANEOUS MESSAGES. In this section, we give lower bounds on the randomized SM complexity of the function $\mathsf{GAF}_{G,k}$ (Theorem 3.3, Lemma 3.4). Up to a constant factor, the bounds match our lower bounds on deterministic SM complexity (Theorem 2.8, Lemma 2.16).

In a *randomized SM protocol* all the players and the referee are allowed to use coin flips.

We consider public coin protocols, i.e., protocols where the random strings used are visible to all parties, including the referee. This is clearly the strongest possible model in the sense that it can simulate at no extra cost the models which allow private or partially private (e.g., hidden from the referee) coins. Therefore any lower bound in the public coin model will automatically remain valid in models with private or partially private coins.

DEFINITION 3.1. *A randomized SM protocol P for a function f is said to have ϵ* advantage *$(0 \le \epsilon \le 1)$ if*

$$\text{for every input } x, \quad \Pr[P(x) = f(x)] - \Pr[P(x) \neq f(x)] \ge \epsilon, \tag{3.1}$$

where the probability is taken over the random choices of the players and the referee.

DEFINITION 3.2. *The* cost *of a randomized SM protocol is the maximum number of bits communicated by any player on any input and for any choice of the random bits. We define the* ϵ-randomized SM complexity of f*, denoted $R_0^\epsilon(f)$, as the minimum cost of a randomized SM protocol for f achieving an advantage $\ge \epsilon$.*

Note that a deterministic protocol has advantage $\epsilon = 1$, so $C_0(f) = R_0^1(f)$.

We also note, for future reference, that inequality (3.1) is equivalent to the following:

$$\Pr[P(x) = f(x)] \ge \frac{1+\epsilon}{2}. \tag{3.2}$$

The main result of this section is a lower bound on the randomized SM complexity of the $\mathsf{GAF}_{G,k}$ function, extending the deterministic lower bound of Theorem 2.8.

THEOREM 3.3. *For any finite group G, and $k \ge 2$,*

$$R_0^\epsilon(\mathsf{GAF}_{G,k}) = \Omega\left(\frac{|G|^{1/(k-1)}\,\epsilon^2}{k-1}\right).$$

This bound will follow from Lemma 3.4 below.

In this and later sections we will express our bounds in terms of the *binary entropy function* H defined as follows:

$$\mathrm{H}(x) := -x\log_2 x - (1-x)\log_2(1-x) \qquad (0 \le x \le 1). \tag{3.3}$$

Note that $\mathrm{H}(0) = \mathrm{H}(1) = 0$. The maximum of H is taken at $x = 1/2$, where $\mathrm{H}(1/2) = 1$.

LEMMA 3.4. *For any finite group G, $0 \le \epsilon \le 1$, and $0 \le \alpha \le 1$,*

$$R_0^\epsilon(\mathsf{GAF}_{G,k}) \ge \frac{\alpha|G|}{(k-1)\rho_\alpha(G,k-1)}(1 - \mathrm{H}(1/2 - \epsilon/2)).$$

This lemma extends the deterministic lower bound of Lemma 2.16. Its proof generalizes the strategy from the deterministic case. While we completely recover x_0 (restricted to the part of the group covered by the decomposition) from all the messages of the players in the proof of Lemma 2.16, here we will be able only to recover "most" bits of an "average" x_0. Lemma 3.7 provides a means to lower bound the amount of information from which such a reconstruction is possible and can be thought of as a generalization of Claim 2.18.

Our main result for this section (Theorem 3.3) is now immediate by combining Corollary 2.20 and Lemma 3.4 and using the following well-known estimate for the binary entropy function:

$$\text{For } |\delta| \le 1/2, \quad 1 - \frac{\pi^2}{3\ln 2}\,\delta^2 \;\le\; \mathrm{H}\left(\frac{1}{2} - \delta\right) \;\le\; 1 - \frac{2}{\ln 2}\,\delta^2. \qquad \square \tag{3.4}$$

Following Yao [Ya83], we prove our lower bound on randomized complexity (Lemma 3.4) via a lower bound on *distributional complexity*.

DEFINITION 3.5. *Given a Boolean function f, a probability distribution μ on its input space, and an ϵ, $0 \le \epsilon \le 1$, a* (μ,ϵ)-SM protocol *for f is a* deterministic *SM protocol P such that*

$$\Pr_{\mu}[P(x) = f(x)] - \Pr_{\mu}[P(x) \ne f(x)] \ge \epsilon,$$

where the probability is with respect to the distribution μ on the input space. We define the (μ,ϵ)-distributional complexity of f, *denoted $C_0^{\mu,\epsilon}(f)$, as the minimum cost of a (μ,ϵ)-SM protocol for f.*

Next we state Yao's key observation, which reduces the question of lower bounds on randomized complexity to lower bounds on distributional complexity with respect to any distribution on inputs.

THEOREM 3.6 (see [Ya83]). *For any function f and any $0 \le \epsilon \le 1$,*

$$R_0^{\epsilon}(f) = \max_{\mu}\; C_0^{\mu,\epsilon}(f).$$

The following lemma provides a tradeoff between the *average* Hamming distance travelled and the number of destinations reached by a transformation of the Boolean cube. The lemma may be of independent interest, in addition to being central to our proof of Lemma 3.4.

LEMMA 3.7 (distance-range tradeoff). *Let $\phi : \{0,1\}^m \longrightarrow \{0,1\}^m$ be a function with range R. Let $0 \le \delta \le 1/2$. Suppose the average Hamming distance between $X \in \{0,1\}^m$ and $\phi(X)$ is $\le \delta m$. Then*

$$|R| \ge 2^{(1-\mathrm{H}(\delta))m}. \tag{3.5}$$

Remark 3.8. Using a random cover of the Boolean cube by Hamming balls one can show that for $\delta < 1/2$, the lower bound (3.5) is optimal within a factor of $c_1(\delta)\sqrt{m}$.

Lemma 3.7 is an immediate consequence of Lemma 3.9 below. $\mathbb{H}(X)$ denotes the entropy of the random variable X. For the concept of entropy of random variables and related facts, we refer to the second edition of Alon and Spencer [AlS00, section 14.6].

LEMMA 3.9 (entropy loss lemma). *Let $\phi : \{0,1\}^m \longrightarrow \{0,1\}^m$. Let $X \in \{0,1\}^m$ be a random element of the domain chosen according to a probability distribution μ. Let $0 \le \delta \le 1/2$. Suppose that*

$$\mathbb{E}[\mathrm{dist}(X, \phi(X))] \le \delta m,$$

where $\mathrm{dist}(\cdot,\cdot)$ *refers to Hamming distance and $\mathbb{E}(\cdot)$ denotes the expected value. Then*

$$\mathbb{H}(X) - \mathbb{H}[\phi(X)] \le \mathrm{H}(\delta)m.$$

Note that if μ is the uniform distribution, then the conditions in Lemmas 3.7 and 3.9 become identical, and $\mathbb{H}(X) = m$. So the conclusion of Lemma 3.7 follows

from the known fact that for any random variable Y with range R, the entropy of Y is bounded by

$$\mathbb{H}[Y] \le \log_2 |R|. \tag{3.6}$$

This completes the proof of Lemma 3.7. □

For the proof of the entropy loss lemma we will use the following elementary facts from information theory [AlS00, section 14.6]:

- For any two random variables U and V,

$$\mathbb{H}[U,V] = \mathbb{H}[V] + \mathbb{H}[U|V]. \tag{3.7}$$

-

$$\mathbb{H}[U|V] \le \mathbb{H}[U]. \tag{3.8}$$

- If V is completely determined by U, then

$$\mathbb{H}[U,V] = \mathbb{H}[U]. \tag{3.9}$$

- Let $X = (X_1, \ldots, X_m)$. Then

$$\mathbb{H}[X] = \mathbb{H}[X_1, \ldots, X_m] \le \sum_{i=1}^{m} \mathbb{H}[X_i]. \tag{3.10}$$

- The binary entropy function H defined in (3.3) is *concave*:

$$\sum_i \alpha_i = 1, \quad 0 \le \alpha_i \le 1 \implies \sum_i \alpha_i \mathrm{H}(p_i) \le \mathrm{H}\left(\sum_i \alpha_i p_i\right). \tag{3.11}$$

Proof of the entropy loss lemma. Let $Y = \phi(X)$. Note that

$$\mathbb{H}[X] - \mathbb{H}[Y] = \mathbb{H}[X,Y] - \mathbb{H}[Y] = \mathbb{H}[X|Y], \tag{3.12}$$

where the first equality follows from (3.9), since Y is completely determined by X, and the second follows from (3.7). So the conclusion of Lemma 3.9 is equivalent to the inequality

$$\mathbb{H}[X|Y] \le m\mathrm{H}(\delta). \tag{3.13}$$

Let $X = (X_1, \ldots, X_m)$ and $Y = (Y_1, \ldots, Y_m)$. For $1 \le i \le m$, let Z_i denote the indicator random variable of the event $X_i \ne Y_i$, i.e.,

$$Z_i := \begin{cases} 1 & \text{if } X_i \ne Y_i, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\delta_i := \Pr[Z_i = 1] = \mathbb{E}[Z_i]$. Then $\sum_i Z_i = \mathrm{dist}(X,Y)$. It follows that

$$\sum_{i=1}^{m} \delta_i = \sum_{i=1}^{m} \mathbb{E}[Z_i] = \mathbb{E}\left[\sum_{i=1}^{m} Z_i\right] = \mathbb{E}[\mathrm{dist}(X,Y)] \le \delta m. \tag{3.14}$$

Claim 3.10. *For $1 \le i \le m$, we have*

$$\mathbb{H}[X_i|Y] \le \mathrm{H}(\delta_i). \tag{3.15}$$

Assuming the validity of the claim, the following string of inequalities yields the bound (3.13):

$$\begin{aligned}
\mathbb{H}[X|Y] &\le \sum_{i=1}^{m} \mathbb{H}[X_i|Y] \qquad \text{using (3.10)} \\
&\le \sum_{i=1}^{m} \mathrm{H}(\delta_i) \qquad \text{by the claim} \\
&\le m\mathrm{H}\left(\frac{1}{m}\sum_{i=1}^{m}\delta_i\right) \qquad \text{by (3.11)} \\
&\le m\mathrm{H}(\delta) \qquad \text{since for } 0 \le x \le 1/2,\ \mathrm{H}(x) \text{ is increasing.}
\end{aligned}$$

Finally, we need to prove Claim 3.10. We use $a \oplus b$ to denote the mod 2 sum of a and b.

$$\begin{aligned}
\mathbb{H}[X_i|Y] &= \mathbb{H}[X_i \oplus Y_i|Y] \quad \text{since for any fixed } y,\ X_i \text{ and } X_i \oplus y_i \text{ have identical entropies} \\
&\le \mathbb{H}[X_i \oplus Y_i] = \mathbb{H}[Z_i] \quad \text{using (3.8) and the definition of } Z_i \\
&= \mathrm{H}(\delta_i) \quad \text{since } Z_i \text{ is a binary random variable with } \Pr[Z_i = 1] = \delta_i.
\end{aligned}$$

This completes the proof of the entropy loss lemma. □

Next we prove our main result.

Proof of Lemma 3.4. We will prove a lower bound on $C_0^{\mu,\epsilon}(\mathsf{GAF}_{G,k})$ for some distribution μ and apply Theorem 3.6.

Let G be a group of order n. Let $H_1, \ldots, H_{k-1}$ be an optimal collection of subsets of G from Definition 2.12, and let $K = H_1 \cdot \ldots \cdot H_{k-1}$. Let $C = H_1 \times \cdots \times H_{k-1}$. Note that $K \subseteq G$ and $C \subseteq G \times \cdots \times G$ ($(k-1)$ times). Note further that $|K| \le |C|$ and multiplication provides a natural onto map $C \to K$. Let $B \subseteq C$ be a set of representatives of the preimages under this map; so $|K| = |B|$, and each element of $g \in K$ can be uniquely factored as $g = h_1 \cdot \ldots \cdot h_{k-1}$, where $(h_1, \ldots, h_{k-1}) \in B$.

The distribution μ we will use is the uniform distribution on $\{0,1\}^K \times B$; i.e., player 0 will be given a uniformly chosen $x_0 : K \longrightarrow \{0,1\}$, and players 1 through $k-1$ will be given a uniformly chosen $(k-1)$-tuple from B. (Strictly speaking, x_0 will be taken from functions $G \longrightarrow \{0,1\}$ that are fixed to zero on $G \setminus K$.)

Given $x_0 \in \{0,1\}^K$, we define $w_{x_0} \in \{0,1\}^{\ell|\widehat{H}_1|} \times \cdots \times \{0,1\}^{\ell|\widehat{H}_{k-1}|}$ as before in the proof of Lemma 2.16 from the players' functions p_i for $1 \le i \le k-1$. The ℓ-bit segment $(w_{x_0})_{\widehat{h}_i}$ of w_{x_0} corresponding to $\widehat{h}_i \in \widehat{H}_i$ is defined as $(w_{x_0})_{\widehat{h}_i} = p_i(x_0, \widehat{h}_i)$. We again define a function F:

$$F : \{0,1\}^{\ell|\widehat{H}_1|} \times \cdots \times \{0,1\}^{\ell|\widehat{H}_{k-1}|} \longrightarrow \{0,1\}^K.$$

The function F is defined as in the proof of Lemma 2.16 from the referee's function r and player 0's function p_0. Specifically, for $g \in K$, let $(h_1, \ldots, h_{k-1}) \in B$ be the unique factorization of g in B. For $w \in \mathrm{domain}(F)$, we set

$$(F(w))(g) = r\left(p_0(h_1, \ldots, h_{k-1}), w_{\widehat{h}_1}, \ldots, w_{\widehat{h}_{k-1}}\right).$$

For $x_0 \in \{0,1\}^K$, define $y_0 := F(w_{x_0}) \in \{0,1\}^K$. We claim that the average Hamming distance between x_0 and y_0 (averaging over $x_0 \in \{0,1\}^K$) is at most $|K|(1-\epsilon)/2$.

Indeed, let us form a $\{0,1\}^K \times B$ $(0,1)$-matrix M as follows: $M(x_0,(h_1,\ldots,h_{k-1}))=1$ if and only if the protocol P makes an error when player 0 has x_0 on her forehead and player i for $1 \le i \le k-1$ has h_i on her forehead, i.e., $x_0(h_1 \cdot \ldots \cdot h_{k-1}) \neq r(p_0(h_1,\ldots,h_{k-1}),p_1(x_0,\widehat{h}_1),\ldots,p_{k-1}(x_0,\widehat{h}_{k-1}))$. By the definition of y_0, we have $M(x_0,(h_1,\ldots,h_{k-1}))=1$ if and only if $x_0(h_1 \cdot \ldots \cdot h_{k-1}) \neq y_0(h_1 \cdot \ldots \cdot h_{k-1})$. Moreover, since the protocol P has ϵ advantage, by inequality (3.2) it follows that the fraction of 1's in M is at most $(1-\epsilon)/2$. Hence the average distance between x_0 and y_0 is at most $|K|(1-\epsilon)/2$.

Now an application of the distance-range tradeoff lemma (Lemma 3.7) concludes the proof as follows. Let $m = |K| = \alpha|G|$. Define $\phi : \{0,1\}^m \to \{0,1\}^m$ by setting $\phi(x_0) = y_0 = F(w_{x_0})$. Let $\delta = (1-\epsilon)/2$. We have just verified the average-distance condition, so Lemma 3.7 implies that the range R of y_0 satisfies $\log|R| \ge |K|\mathrm{H}(1/2-\epsilon/2)$. On the other hand, the range of y_0 is not larger than the domain of F:

$$(k-1)\ell\rho_\alpha(G,k-1) \ge \log|R| \ge \alpha|G|(1-\mathrm{H}(1/2-\epsilon/2)),$$

and hence

$$C_0^{\mu,\epsilon}(\mathsf{GAF}_{G,k}) = \ell \ge \frac{\alpha|G|}{(k-1)\rho_\alpha(G,k-1)}\left(1-\mathrm{H}(1/2-\epsilon/2)\right).$$

This completes the proof of Lemma 3.4. □

This also completes the proof of the main results of this section. The rest of the section is devoted to a discussion of the need to use information theory (entropy) arguments and to clarifying the connections with the papers [BJKS02] and [BaKL95].

Remark 3.11. Our central "entropy" tool was Lemma 3.7; its proof was the only place where the concept of entropy was used. The question arises whether the use of entropy was necessary at all.

The key word in Lemma 3.7 is "average." If we impose the bound $\mathrm{dist}(X,\phi(X)) \le \delta m$ on *all* $X \in \{0,1\}^m$, then the conclusion will follow from a straightforward Hamming bound on coverings of the Boolean cube. Indeed, in this case the Hamming balls of radius δm about R would cover the entire Boolean cube; therefore

$$2^m \le |R| \sum_{k \le \delta m} \binom{m}{k} \le |R| 2^{m\mathrm{H}(\delta)}, \tag{3.16}$$

proving the desired inequality. In the last step we used the bound $\sum_{k\le\delta m}\binom{m}{k} \le 2^{m\mathrm{H}(\delta)}$, which is valid for all m and δ $(0 \le \delta \le 1/2)$ (see, e.g., [MacS77, p. 310]). (For $\delta < 1/2$ and large m, the bound can be improved by a factor of $\sim c(\delta)/\sqrt{m}$; cf. (5.1).)

More significantly, even if the condition is on *average* distance, the use of entropy can be avoided to obtain a slightly weaker result by a Markov inequality argument combined with the Hamming bound indicated above.

Indeed, under the conditions of Lemma 3.7 one can prove, without the use of entropies, the following lower bound on $|R|$ for any constant $c > 0$:

$$|R| \ge \frac{c}{\delta+c} 2^{(1-\mathrm{H}(\delta+c))m}. \tag{3.17}$$

To see (3.17), note that by Markov's inequality on nonnegative random variables, there exists a subset $S \subseteq \{0,1\}^m$ such that $|S| \ge 2^m c/(\delta+c)$ and for *all* $X \in S$,

$\mathrm{dist}(X, \phi(X)) \leq (\delta + c)m$. Now apply the Hamming bound argument as in (3.16) above to get a lower bound on $|\phi(S)|$, and hence on $|R|$.

Furthermore, an application of this weaker inequality would essentially suffice for a proof of the main result of this section, the lower bound for R_0^ϵ (Lemma 3.4).

To deduce a lower bound on R_0^ϵ from inequality (3.17), we can choose $c = \epsilon/4$ and note that (cf. proof of Lemma 3.4) $\delta = (1-\epsilon)/2$. This implies $c/(\delta + c) \geq \epsilon/2$ and leads to a bound only slightly weaker than Lemma 3.4:

$$R_0^\epsilon(\mathsf{GAF}_{G,k}) \geq \frac{\alpha|G|(1 - \mathrm{H}(1/2 - \epsilon/4)) + \log(\epsilon/2)}{(k-1)\rho_\alpha(G, k-1)}.$$

Using this inequality we obtain the lower bound

$$R_0^\epsilon(\mathsf{GAF}_{G,k}) = \Omega\left(\frac{|G|^{1/(k-1)}\ \epsilon^2 + \log\epsilon}{k-1}\right),$$

which is only slightly weaker than the lower bound on R_0^ϵ given in Theorem 3.3.

The conclusion is that, in essence, entropy arguments are not needed for the main results of this section. On the other hand, our simple and elegant entropy argument makes the conclusions also more elegant.

Remark 3.12. A key step in our entropy argument is Claim 3.10. We note that the claim is in fact "Fano's inequality" [CT91] for the special case of Boolean variables.

First, we state Fano's inequality on the prediction errors for Boolean variables.

PROPOSITION 3.13 (Fano's inequality for Boolean variables). *Let X be a Boolean random variable and Y a random variable over the domain S_Y. Let $g : S_Y \to \{0,1\}$ be a "prediction function" (given the value of $Y \in S_Y$, g guesses the value of X). Let δ be the "prediction error": $\delta = \Pr[g(Y) \neq X]$. Then $\mathbb{H}[X \mid Y] \leq \mathrm{H}(\delta)$.*

Claim 3.10 follows from Proposition 3.13 as follows.

For $1 \leq i \leq m$, let us define $g_i : \{0,1\}^m \to \{0,1\}$ by setting $g_i(Y) = Y_i$. Let us use g_i to predict X_i given Y. The prediction error is $\delta_i = \Pr[X_i \neq Y_i]$. Fano's inequality gives $\mathbb{H}[X_i|Y] \leq \mathrm{H}(\delta_i)$, which is exactly inequality (3.15), completing the proof of Claim 3.10. □

Conversely, our proof of Claim 3.10 in effect proves Proposition 3.13. Indeed, our proof of Claim 3.10 can be found in the last three lines of the proof of the entropy loss lemma above. To see how to adapt those three lines to prove Proposition 3.13, replace X_i by X, Y_i by $g(Y)$, Z_i by $Z := X \oplus g(Y)$, and δ_i by δ. This completes the proof of Fano's inequality. □

Remark 3.14. (comparison with [BJKS02] and [BaKL95]) Independent of our work, Bar-Yossef et al. [BJKS02] describe an information-theoretic approach to proving lower bounds on the distributional SM complexity of $\mathsf{GAF}_{G,k}$ analogous to our Lemma 3.4.

The [BJKS02] result differs from ours in their definition of the ρ parameter (based, apparently, on an optimistic interpretation of the ρ parameter defined in the original [BaKL95] paper).

The [BJKS02] definition of ρ assumes a decomposition of the *entire* group G as a product of *subgroups*. These assumptions apparently lead to large values of ρ and thus to poor estimates of the complexity. [BJKS02] make no attempt to estimate the value of their ρ parameter.

The [BaKL95] definition of ρ used *subsets* rather than subgroups of G as factors in the decomposition of the entire group G. It is shown in [BaKL95] that this approach gives a bound for every group of order n which is only slightly worse than the bound

obtained for the "nicest" groups (cyclic and elementary abelian groups), namely, by a factor of $O(\sqrt{\log n})$. A positive outcome of the "Modified Rohrbach Problem (section 7)" would eliminate this factor.

In the present paper we eliminate this factor in a different way, by bypassing the obstacle posed by the Rohrbach problem. In section 2.3 we have constructed optimal (up to a constant factor) decompositions of a *positive fraction* of G into a product of *subsets* (Theorem 2.19). This approach yields SM lower bounds for *all groups* that are within a constant factor of the results for the "nicest" groups.

The foregoing comments apply to the deterministic lower bound. The actual contribution of [BJKS02] is an information-theoretic argument to extend the proof of the deterministic lower bound to distributional complexity. Specifically, [BJKS02] uses "Fano's inequality for Boolean variables" on prediction errors in terms of conditional entropy (see above in Proposition 3.13).

The information-theoretic arguments presented in [BJKS02] remain valid in the context of the more general decompositions considered in our paper which correspond to the ρ_α parameter defined in Definition 2.12.

[BJKS02] use their entropy argument to prove their analogue of Lemma 3.4. Although our proof of Lemma 3.4 is also entropy-based, the two proofs look rather different. Our attempt to find the "common core" of the two proofs yielded only a modest result (see Remark 3.12).

4. Applications to lower bounds in circuit complexity. In this section, we derive some consequences of the SM lower bounds from section 2.2 to superpolynomial lower bounds on certain depth-2 circuits. These circuits are described by the following definition.

DEFINITION 4.1. *A* (SYM,AND)*-circuit is defined to be a depth-*2 *circuit with a symmetric gate at the top and AND gates at the bottom. (We draw circuits with the output at the top. Hence inputs to the bottom gates are input variables and their negations).*

We note that Beigel and Tarui [BeT91] reduce ACC circuits to (SYM,AND)-circuits of quasi-polynomial size with bottom fan-in polylog(n). We present below a lower bound of $\exp((\log n/\log\log n)^2)$ on the size of (SYM,AND)-circuits (of arbitrary bottom fan-in) computing some very weak functions. In fact, our lower bound applies to a function in ACC that contains $\mathsf{GAF}_{\mathbb{Z}_2^t,k}$ as a subfunction.

The following remarkable observation by Håstad and Goldmann relates multiparty communication complexity to certain depth-2 circuits.

LEMMA 4.2 (see [HG91]). *Suppose a function f is computed by a depth-*2 *circuit consisting of an arbitrary symmetric gate of fan-in s at the top and bottom gates computing* arbitrary functions of at most $k-1$ variables*. Then, for any partition of the input among the players, the k-party communication complexity of f is $O(k\log s)$.*

For completeness, we give a proof of Lemma 4.2.

Proof. Since each bottom gate of the circuit has fan-in at most $k-1$, there is at least one player who can evaluate that gate. Partition the bottom gates among the players such that all the gates assigned to a player can be evaluated by that player. Now each player broadcasts the number of her gates that evaluate to 1. This takes $O(\log s)$ bits per player since the top gate has fan-in at most s. Now one of the players can add up all the numbers broadcasted to compute the symmetric function given by the top gate and announce the value of the function. □

It is obvious that this proof works in the SM model as well: each player sends to the referee the number of gates evaluating to 1 among his gates, and the referee adds

these numbers to compute f. The SM complexity of the protocol is clearly $O(\log s)$. Hence, we get the following corollary.

COROLLARY 4.3. *Suppose a function f is computed by a depth-2 circuit consisting of an arbitrary symmetric gate of fan-in s at the top and bottom gates computing* arbitrary functions of at most $k-1$ variables. *Then, for any partition of the input among the players, the k-party SM complexity of f is $O(\log s)$.*

This observation, pointed out to us by Avi Wigderson, serves as the main motivation for considering SM complexity.

The next lemma uses the method of random restrictions [Aj83, FSS84] to reduce the fan-in of the bottom gates and at the same time to ensure that the "target function" (with high SM complexity) is computed by the restricted circuit. We note that a similar technique is used by Razborov and Wigderson [RaW93].

First, we introduce a definition.

DEFINITION 4.4. *Let $f(x_1,\ldots,x_v)$ be a Boolean function of v variables. Let $S_1,\ldots,S_v$ be disjoint sets of variables, where $|S_i| = b$ for all i. Then the* Parity_b $\mathsf{Blow\text{-}up}$ *of f is the function g on vb variables defined by*

$$g(y_1,\ldots,y_{vb}) = f(\oplus_{i\in S_1} y_i,\ldots,\oplus_{i\in S_v} y_i).$$

DEFINITION 4.5. *For a set X of Boolean variables a* restriction *ρ is defined to be a mapping $\rho : X \longrightarrow \{0,1,*\}$. We interpret a variable assigned a $*$ to be "free," i.e., not fixed to a constant. Given a function f on X, its restriction $f_{|\rho}$ is the induced function on variables assigned a $*$ by ρ obtained by evaluating f when the nonfree variables are fixed according to ρ. For a circuit C, its restriction $C_{|\rho}$ is the circuit (with variables from $\rho^{-1}(*)$) obtained by fixing the variables of C assigned 0 or 1 by ρ and simplifying wherever possible.*

LEMMA 4.6. *Let g be the* Parity_b $\mathsf{Blow\text{-}up}$ *of f. Let ℓ be a parameter satisfying $\ell \le \log b - \log\ln v + 1$, and let $0 < c < 1$ be a constant. Suppose that g is computed by a circuit C consisting of at most $2^{c\cdot\ell^2}$ AND gates at the bottom. Then there is a restriction ρ such that*

- *all AND gates at the bottom level of $C_{|\rho}$ have fan-in at most ℓ,*
- *$C_{|\rho}$ has v input variables, exactly one from each block S_i, and*
- *$C_{|\rho}$ computes f.*

Proof. We define ρ in two stages. First, we obtain a random restriction ρ_1 that reduces the fan-in of each bottom AND gate to at most ℓ and keeps alive at least two variables from each block S_i. We prove the existence of ρ_1 below. Second, we define ρ_2 by assigning values to all but one of the variables from each S_i left alive by ρ_1 so that we are left with exactly one *unnegated* variable from each S_i; i.e., $\rho_2(\rho_1(\oplus_{j\in S_i} y_j)) = y_{i'}$ for some $y_{i'} \in S_i$. The desired restriction ρ is the composition of ρ_1 and ρ_2. Moreover, by the definition of g, the restricted circuit computes $f(y_{1'},\ldots,y_{v'})$.

Let $p := (2\ln v)/b$. Note that $p \le 2^{-\ell}$. We choose ρ_1 by independently assigning to each variable a $*$ (to keep it alive) with probability p and a 0 or 1, each with probability $(1-p)/2$. Let γ be a bottom level AND gate of C, and let m be the fan-in of γ.

First consider the case when $m \le \ell^2$. Without loss of generality, $m \ge \ell$. Then

$$\begin{aligned}\Pr[\gamma_{|\rho_1} \text{ has fan-in } > \ell] &\le \sum_{i>\ell} \binom{m}{i} p^i \left(\frac{1-p}{2}\right)^{m-i} \\ &\le (1+o(1))\binom{m}{\ell} p^\ell \left(\frac{1-p}{2}\right)^{m-\ell} \qquad \text{since } \ell \gg mp\end{aligned}$$

$$\leq (1+o(1))(pe\ell)^{\ell} \quad \text{since } m \leq l^2$$
$$\leq 2^{-\ell^2\cdot(1-o(1))} \quad \text{since } p \leq 2^{-\ell}.$$

Next consider the case when $m > \ell^2$. Then

$$\Pr[\gamma_{|\rho_1} \text{ has fan-in } > \ell] \leq \Pr[\gamma_{|\rho_1} \not\equiv 0] \leq \left(\frac{1+p}{2}\right)^m$$
$$\leq 2^{-\ell^2\cdot(1-o(1))} \quad \text{since } m > \ell^2 \text{ and } p \leq 2^{-\ell}.$$

Since C has at most $2^{c\cdot\ell^2}$ AND gates at the bottom (where $c < 1$ is a constant), from the preceding two cases it follows that

$$\Pr[\text{ some bottom AND gate of } C_{|\rho_1} \text{ has fan-in} > l] = o(1). \tag{4.1}$$

Moreover, for a fixed i, $1 \leq i \leq v$, we have

$$\Pr[\rho_1(S_i) \text{ has} < 2 *\text{'s}] = (1-p)^b + bp(1-p)^{b-1}$$
$$\leq e^{-pb}(1+bp/(1-p))$$
$$\leq O(\log v/v^2) \quad \text{since } p \geq 2\ln v/b.$$

Hence, we also have

$$\Pr[\ \rho_1 \text{ assigns fewer than 2 } *\text{'s to some block } S_i\ ] = o(1). \tag{4.2}$$

From (4.1) and (4.2), we see that with high probability all bottom AND gates of $C_{|\rho_1}$ have fan-in at most ℓ, and, furthermore, the inputs of $C_{|\rho_1}$ will have at least two variables from each block S_i. Hence such a restriction ρ_1 exists.

By composing ρ_1 with an additional restriction ρ_2 as described in the beginning of the proof, we complete the proof of the lemma. □

THEOREM 4.7. *Suppose an n variable function f has k-party SM complexity at least $c_0(n,k)$ for some partition of the input among the players. Then any* (SYM,AND)*-circuit computing the* Parity_n $\mathsf{Blow\text{-}up}$ *of f must have size at least* $\min\{\exp(k^2), \exp(c_0(n,k))\}$.

Proof. Let g denote the Parity_n $\mathsf{Blow\text{-}up}$ of f, and let C be a minimal size (SYM,AND)-circuit computing g. If C has size $> 2^{(k-1)^2}$, we are done.

Therefore, suppose size of C is $\leq 2^{(k-1)^2}$. We apply Lemma 4.6 to obtain a restriction ρ such that bottom gates of $C_{|\rho}$ have fan-in at most $k-1$ and $C_{|\rho}$ computes f.

Now applying Corollary 4.3, we see that the size of $C_{|\rho}$ must be exponential in the SM complexity $c_0(n,k)$ of f. Hence C itself must have size $\geq \exp(c_0(n,k))$. □

Using our lower bound on SM complexity from section 2 we immediately get the following corollary.

COROLLARY 4.8. *Let G be any group of order n, and let $k = \epsilon \log n/\log\log n$ for a sufficiently small constant $\epsilon > 0$. Then any* (SYM,AND)*-circuit computing the* Parity_n $\mathsf{Blow\text{-}up}$ *of* $\mathsf{GAF}_{G,k}$ *must have size* $\exp((\log n/\log\log n)^2)$.

Proof. From Theorem 2.8, we have that for $c_0(n,k) := C_0(\mathsf{GAF}_{G,k}) = \Omega(n^{1/(k-1)}/(k-1))$. Hence, if $k \leq \epsilon \log n/\log\log n$ for sufficiently small $\epsilon > 0$, $c_0(n,k) \geq k^2$. Now we get the claimed bound from Theorem 4.7. □

5. A Simultaneous Messages protocol for $\mathsf{GAF}_{\mathbb{Z}_2^t,k}$. In this section we give a nontrivial protocol for $\mathsf{GAF}_{G,k}$ for $G = \mathbb{Z}_2^t$. Our protocol yields an upper bound of about $n^{(\log k)/k}$, where $n = 2^t$ (see Theorem 5.5). In particular, for three players we obtain a nontrivial $O(n^{0.92})$ upper bound (see Theorem 5.3). These upper bounds have been subsequently improved in [AmL00], giving in particular an upper bound of $O(n^{0.73})$ for three players. These upper bounds should be compared with our lower bound of $\Omega(n^{1/(k-1)}/(k-1))$ (Theorem 2.8).

It is curious to remark that, in contrast to the lower bound, our protocol heavily depends on the specific structure of this group. In particular, it does not apply to the cyclic group $G = \mathbb{Z}_n$. For cyclic groups, Pudlák, Rödl, and Sgall [PRS97] give upper bounds of $O(n(\log\log n/\log n)^k)$, for constant k, and $O(n^{6/7})$ for $k \geq c\log n$ for some constant c. These upper bounds have been significantly improved by Ambainis [Am96] to $O(\frac{n\log^{1/4} n}{2^{\sqrt{\log n}}})$ for $k = 3$ and to $O(n^\epsilon)$ for an arbitrary $\epsilon > 0$ for $k = O((\log n)^{c(\epsilon)})$. However, the bounds for $\mathbb{Z}_n$ are still much weaker than the corresponding bounds for $\mathbb{Z}_2^t$ presented in this paper.

We will think of the n-bit string A held by p_0 (previously denoted x_0) as a Boolean function on $t := \log n$ variables $z_1, \ldots, z_t$, i.e., $A : \{0,1\}^t \longrightarrow \{0,1\}$. For $1 \leq i \leq k-1$, let x_i be the t-bit string held by player p_i. Then we have

$$\mathsf{GAF}_{\mathbb{Z}_2^t,k}(A, x_1, \ldots, x_{k-1}) = A(x_1 + \cdots + x_{k-1}),$$

where "+" denotes addition in $\mathbb{Z}_2^t$.

5.1. Three players. We will first describe the protocol for three players. The idea extends naturally to the general case. For simplicity of notation, let p_1 hold x and p_2 hold y. At the cost of $2t$ bits, p_0 sends the strings x and y to the referee. Since this communication will be insignificant, we can henceforth ignore p_0 and assume that the referee knows x and y (but not A). Then we want to minimize the number of bits sent by p_1 and p_2 that will enable the referee to compute $A(x+y)$.

The protocol will be based on the fact that the Boolean function A can be represented as a multilinear polynomial of (total) degree at most t. In fact, the following lemma is the crucial observation in the protocol. We use the notation

$$\Lambda(m,b) = \sum_{j=0}^{b} \binom{m}{j}$$

and the fact that for fixed δ, $0 < \delta < 1/2$,

$$\Lambda(m, \delta m) \sim c(\delta) 2^{m\mathrm{H}(\delta)}/\sqrt{m}. \tag{5.1}$$

Lemma 5.1. *Given the promise that A is a multilinear polynomial of degree d over $\mathbb{Z}_2$, $x, y \in \mathbb{Z}_2^t$, $\mathsf{GAF}_{\mathbb{Z}_2^t,3}(A,x,y)$ has an SM protocol with cost $\Lambda(t, \lfloor d/2 \rfloor)$.*

Proof. Let A be given by $A(z) = \sum_{S\subseteq[t],\,|S|\leq d} a_S Z_S$, where Z_S denotes the monomial $\prod_{i\in S} z_i$. Thus,

$$A(x+y) = \sum_{|S|\leq d} a_S \prod_{i\in S}(x_i + y_i) = \sum_{|S|\leq d} a_S \sum_{T_1 \dot\cup T_2 = S} X_{T_1} Y_{T_2},$$

where $X_{T_1} := \prod_{i\in T_1} x_i$ and $Y_{T_2} := \prod_{i\in T_2} y_i$.

We can rewrite this as follows:

$$A(x+y) = \sum_{|T_1|\le\lfloor d/2\rfloor} \left(\sum_{|T_1\dot\cup T_2|\le d} a_{T_1\dot\cup T_2}\, Y_{T_2} \right) \cdot X_{T_1} + \sum_{|T_2|\le\lfloor d/2\rfloor} \left(\sum_{|T_1\dot\cup T_2|\le d} a_{T_1\dot\cup T_2}\, X_{T_1} \right) \cdot Y_{T_2},$$

where T_1 and T_2 are disjoint subsets of $[t]$, and we assume without loss of generality that terms $a_{T_1\dot\cup T_2}\, X_{T_1}\, Y_{T_2}$ with both $|T_1|$ and $|T_2|$ less than or equal to $\lfloor d/2\rfloor$ are placed in the first sum.

We now observe that the first sum in the last equation is a polynomial in x whose coefficients (which depend only on $a_{T_1\dot\cup T_2}\, Y_{T_2}$) are known to p_1. Similarly, the second sum is a polynomial in y whose coefficients are known to p_2. The degree of both polynomials is bounded by $\lfloor d/2\rfloor$. Hence, using at most $\Lambda(t, \lfloor d/2\rfloor)$ bits, each player can communicate the coefficients of their corresponding polynomial to the referee. Since the referee already knows x and y, he can evaluate the two polynomials and add them up to announce the value of $A(x+y)$. Since p_0 used only $2t$ bits to send x and y to the referee, the cost of the protocol is simply $\Lambda(t, \lfloor d/2\rfloor)$. □

Remark 5.2. For small enough d, the protocol is a quadratic improvement over the trivial one where the entire function A is communicated to the referee.

Suppose now that A is an arbitrary Boolean function. We will use Lemma 5.1 on the low-degree part of A and the trivial protocol on the high-degree part. Since there are not too many high-degree terms, we will be able to keep the communication within n^c for some $c < 1$.

THEOREM 5.3. $C_0(\mathsf{GAF}_{\mathbb{Z}_2^t,3}) = o(n^{0.92})$.

Proof. Let $A : \mathbb{Z}_2^t \longrightarrow \{0,1\}$ and $x, y \in \mathbb{Z}_2^t$ be the inputs on the foreheads of players 0, 1, and 2, respectively. Write A as a multilinear polynomial over $\mathbb{Z}_2$ of degree at most t: $A(z) = \sum_{S\subseteq[t]} a_S\, Z_S$. Define A' to be the part of A corresponding to degree less than or equal to $2t/3$, and let A'' be the remaining part of A. That is,

$$A(z) = \sum_{|S|\le 2t/3} a_S\, Z_S \;+\; \sum_{|S|>2t/3} a_S\, Z_S = A'(z) + A''(z).$$

Players p_1 and p_2 use the protocol of Lemma 5.1 on A'. They just send the high-degree terms a_S for $|S| > 2t/3$ directly to the referee (each sends half of them). The number of bits used by each of p_1 and p_2 is at most

$$\Lambda(t, t/3) + \frac{1}{2} \sum_{j>2t/3} \binom{t}{j} \le \frac{3}{2}\Lambda(t, t/3) \le O(2^{t\,\mathrm{H}(1/3)}/\sqrt{t}),$$

using the estimate (5.1). Since p_0 sends fewer bits than p_1 or p_2 (p_0 sends only $2t$ bits), the protocol has cost $O(n^{\mathrm{H}(1/3)}/\sqrt{\log n})$. As $\mathrm{H}(1/3) = 0.91829\ldots$, the theorem follows. □

5.2. *k* players. We generalize the idea from the preceding section to k players. An extension of Lemma 5.1 follows.

LEMMA 5.4. *Given the promise that A is a t-variable multilinear polynomial of degree d over $\mathbb{Z}_2$, and for $1 \le i \le k-1$, $x_i \in \mathbb{Z}_2^t$, $\mathsf{GAF}_{\mathbb{Z}_2^t,k}(A, x_1, \ldots, x_{k-1})$ has an SM protocol with cost at most $\Lambda(t, \lfloor d/(k-1)\rfloor) + t$.*

Proof. In the 3-player protocol, player p_0 passes the two short inputs. In the k-player protocol, the task of passing the short inputs $x_1, \ldots, x_{k-1}$ will be divided

among players p_1 through p_{k-1}, and p_0 will remain silent. This avoids having one player (p_0) communicate too many bits in case k is large.

Letting $A(z) = \sum_{S\subseteq[t],\,|S|\le d} a_S\, Z_S$, we have

$$\begin{aligned} A(x_1 + \cdots + x_{k-1}) &= \sum_{|S|\le d} a_S \prod_{j\in S} (x_{1,j} + \cdots + x_{k-1,j}) \\ &= \sum_{|S|\le d} a_S \sum_{T_1 \dot\cup \cdots \dot\cup T_{k-1} = S} X_{1,T_1} \cdots X_{k-1,T_{k-1}}, \\ &\qquad\qquad \text{where } X_{i,T_i} = \prod_{j\in T_i} x_{ij}. \end{aligned}$$

Let us consider a monomial $a_S\, Z_S$. Since the T_i are disjoint, $\sum_{i=1}^{k-1} |T_i| = |S| \le d$, and hence the smallest T_i is of size at most $\lfloor d/(k-1)\rfloor$. Thus in the expansion

$$\sum_{T_1 \dot\cup \cdots \dot\cup T_{k-1} = S} a_S\, X_{1,T_1} \cdots X_{k-1,T_{k-1}},$$

each term can be "owned" by a player p_i such that T_i is the smallest set in that term. (In case of ties, take the smallest such i.) As a result, the value of this monomial on $x_1 + \cdots + x_{k-1}$ can be distributed among the k players by giving them each a polynomial of degree at most $\lfloor |S|/(k-1)\rfloor$.

The proof follows by linearity and proceeds similarly to that of Lemma 5.1. We conclude that for $1 \le i \le k-1$, player p_i needs to send $\Lambda(t, \lfloor d/(k-1)\rfloor)$ bits for the terms they own and t bits to send x_{i+1} (player p_{k-1} sends x_1). □

THEOREM 5.5. $C_0(\mathsf{GAF}_{\mathbb{Z}_2^t,k}) \le \frac{k}{k-1}\Lambda(t, \lfloor t/k\rfloor) + t$.

Proof. Let $A : \mathbb{Z}_2^t \longrightarrow \{0,1\}$ be player 0's input, and let $x_i \in \mathbb{Z}_2^t$ be player i's input for $1 \le i \le k-1$. The proof is similar to the proof of Theorem 5.3: separate the low-degree and high-degree parts of A as follows. Monomials of A of degree higher than $t(1-1/k)$ are sent directly to the referee. (For simplicity, we ignore floors and ceilings in this proof.) By dividing these high-degree monomials equally among the $k-1$ players, we see that each player sends at most $\frac{1}{k-1}\sum_{t(1-1/k)\le i\le t} \binom{t}{i} = \Lambda(t, t/k)/(k-1)$ bits. The remaining low-degree part of A has degree at most $t(1-1/k)$. Applying Lemma 5.4 with $d = t(1-1/k)$, each player sends at most $\Lambda(t, t/k) + t$ bits to handle the low-degree part and to transmit x_{i+1} (player p_{k-1} transmits x_1). Adding up, we get that each player sends at most $\frac{k}{k-1}\Lambda(t, t/k) + t$ bits to the referee. □

COROLLARY 5.6. *For* $3 \le k \le \log n$, $C_0(\mathsf{GAF}_{\mathbb{Z}_2^t,k}) \le n^{O((\log k)/k)} + \log n$.

Proof. Use estimate (5.1) and note that, for $k \ge 3$, $\mathrm{H}(1/k) \le \log(ek)/k$. □

Remark 5.7. It follows that if $k \ge c \log n$ for any constant $c > 0$, then each player sends at most polylog(n) bits to the referee. Moreover, it is easy to see from the proof of Lemma 5.4 that if $k = \log n + 1$, each player sends at most $2 + \log n$ bits and if $k > \log n + 1$, each player needs to send only at most $1 + \log n$ bits.

6. SM upper bounds for other functions. In this section, we give nontrivial upper bounds on the SM complexity of a class of functions defined by certain depth-2 circuits and for a partition of the input variables in which each of the k players misses one input bit from each bottom gate. The n bottom gates are identical with fan-in k and compute certain symmetric functions called symmetric *compressible* functions (see Definition 6.1). The top gate is an arbitrary symmetric gate. We call this class of communication problems the SYMCOM(n,k) problems (see Definition 6.7). It

includes, for example, Majority of Majorities, Generalized Inner Product, and Parity of Thresholds.

We start by defining the functions that are allowed on the bottom level, i.e., the *compressible* functions. Let $X = \{x_1, \ldots, x_k\}$ be a set of Boolean variables and $f(x_1, \ldots, x_k)$ a Boolean function. Alice sees a subset $A \subseteq X$ of the variables, and Bob sees the remaining set $B = X \setminus A$. Consider the one-way communication model where Alice sends a message to Bob, and Bob must deduce the value of f from Alice's message and the part of the input he sees. For a given partition of the set of variables $A \dot\cup B = X$, we denote by $C_{A\to B}(f)$ the minimum number of bits that Alice must send.

DEFINITION 6.1. *A class of Boolean functions $\mathcal{F}$ is called* compressible *if for any partition $A, B \neq \emptyset$, $A \dot\cup B = X$, and any $f \in \mathcal{F}$ we have $C_{A\to B}(f) = O(\log |B|)$.*

Remark 6.2. We refer to a function f as a compressible function if it belongs to a compressible class of functions. The constant implied by the O notation may depend on the class but not on the particular function.

We shall be interested only in compressible *symmetric* functions. Note that in this case, it is clear that $C_{A\to B} \leq \log(|A| + 1)$. The point of our definition is that we require $C_{A\to B} \leq \log |B|$ even when $|B|$ is very small compared to $|A|$. Indeed, we shall use this property for $|B| = \Theta(\log k)$.

EXAMPLE 6.3. *The following Boolean functions are compressible:*

1. $\mathsf{Parity}(x_1, \ldots, x_k) = x_1 \oplus \cdots \oplus x_k$.
2. $\mathsf{Mod}_{m,T}(x_1, \ldots, x_k) = 1$ *if and only if* $\sum_{i=1}^k x_i \in T \pmod m$.
3. $\mathsf{Th}_t^k(x_1, \ldots, x_k) = 1$ *if and only if* $\sum_{i=1}^k x_i \geq t$.

Remark 6.4. We will give an example of a function which is not compressible in section 6.1 below (see Definition 6.12 and Corollary 6.15).

PROPOSITION 6.5. *For every partition $A \dot\cup B = X = \{x_1, \ldots x_k\}$ $(A, B \neq \emptyset)$ we have the following:*

1. $C_{A\to B}(\mathsf{Parity}) \leq 1$.
2. $C_{A\to B}(\mathsf{Mod}_{m,T}) \leq \lceil \log m \rceil$.
3. $C_{A\to B}(\mathsf{Th}_t^k) \leq \lceil \log(|B| + 2) \rceil$.

Proof. The statements about the Parity and Mod_m functions are trivial. We prove the statement of the proposition for threshold functions. Let $A, B \neq \emptyset$, $A \dot\cup B = X = \{x_1, \ldots, x_k\}$ be some partition of the input variables. Let ℓ be the number of 1's in part A. If $\ell < t - |B|$, the value of Th_t^k must be 0, and if $\ell \geq t$ the result must be 1, regardless of part B of the input. For $\ell = t - i$, $i = 1, \ldots, |B|$, the value of the function depends on part $|B|$. Therefore, for Bob to compute the value of f, it is enough for Alice to send one of $|B| + 2$ messages to Bob: one for the case where $\ell < t - |B|$, one for the case where $\ell \geq t$, and one for each value of ℓ from $t - |B|$ to $t - 1$. This requires only $\lceil \log(|B| + 2) \rceil$ bits. □

Let $f : \{0,1\}^k \to \{0,1\}$ and $g : \{0,1\}^n \to \{0,1\}$ be Boolean functions. Consider the following depth-2 circuit. The first level has n f-gates whose inputs are disjoint, so there are nk input bits to the circuit. The second level consists of a g-gate which takes the outputs of the f-gates as its n inputs. We denote the function computed by this circuit by $g \circ f$ (see Figure 6.1).

DEFINITION 6.6. *We define the* (g, f)-communication problem *as the problem of k players computing $g \circ f$, where k is the same number as the fan-in of the f-gates, and the input variables are partitioned among the k players so that the jth player misses the jth input bit of each f-gate.*

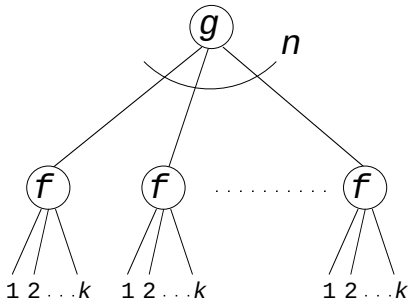


FIG. 6.1. *The depth-2 circuit defining $g \circ f$.*

DEFINITION 6.7. *We call the (g, f)-communication problem a* SYMCOM(n, k) PROBLEM *if the function g is symmetric and the function f is compressible and symmetric.*

Remark 6.8. Observe that the AND function is compressible (cf. Proposition 6.5(3)). Hence the circuits used to define a SYMCOM(n, k) communication problem above are somewhat similar to the (SYM,AND)-circuits defined in section 4. However, there are some crucial differences. First, inputs to distinct bottom gates of circuits in this section (Figure 6.1) are required to be *disjoint*, whereas no such restriction is imposed in (SYM,AND)-circuits. Second, the number of players in this section is *equal* to the bottom fan-in of the circuits, and this is not necessarily true of (SYM,AND)-circuits.

In Theorem 6.11, we will show that there are efficient SM protocols for SYMCOM(n, k) problems for $k > 1 + \log n$ players. First, we need two lemmas.

LEMMA 6.9. *Let t and n be positive integers such that $t > 1 + \log n$. Let $b_0, \ldots, b_{t-1}$ be integers. Consider the following system of t equations in $t + 1$ unknowns:*

$$(t-i)y_i + (i+1)y_{i+1} = b_i, \quad i = 0, 1, \ldots, t-1. \tag{6.1}$$

Assume further that

$$y_i \geq 0, \quad i = 0, 1, \ldots, t; \quad \sum_{i=0}^{t} y_i \leq n. \tag{6.2}$$

Then, under constraints (6.2)*, the system of equations* (6.1) *has at most one integral solution.*

Proof. Let $y = (y_0, \ldots, y_t)$ and $y' = (y'_0, \ldots, y'_t)$ be two solutions of (6.1), each consisting of nonnegative integers whose sum is at most n. For $i = 0, 1, \ldots, t$, let $d_i = y_i - y'_i$. Since $y \neq y'$, we know there exists at least one $d_i \neq 0$.

From (6.1), we obtain the following equations:

$$(t-i)d_i + (i+1)d_{i+1} = 0, \quad i = 0, 1, \ldots, t-1. \tag{6.3}$$

From the $i = 0$ equation, we can express d_1 in terms of d_0: $d_1 = -td_0 = -\binom{t}{1}d_0$. From this and the $i = 1$ equation, we get

$$d_2 = \frac{-(t-1)}{2}d_1 = \frac{(t-1)t}{2}d_0 = \binom{t}{2}d_0.$$

Continuing in this way, we see that for $i = 0$ to t, $d_i = (-1)^i\binom{t}{i}d_0$. Since some d_i is not 0, we know that $d_0 \neq 0$. Furthermore, since the d_i are integers, we know that $|d_0| \geq 1$, and thus $|d_i| \geq \binom{t}{i}$. We also note that since $y_i, y_i' \geq 0$, we have

$$y_i + y_i' \geq |y_i - y_i'| = |d_i|.$$

Now we use the fact that the sum of the y_i and the sum of the y_i' are both at most n to derive a contradiction and complete the proof:

$$2n \geq \sum_{i=0}^{t}(y_i + y_i') \geq \sum_{i=0}^{t}|d_i| \geq \sum_{i=0}^{t}\binom{t}{i} = 2^t > 2^{1+\log n} = 2n. \qquad \square$$

LEMMA 6.10. *Let n be a positive integer, and let M be a $t \times m$ $(0,1)$-matrix, with $m \leq n$ and $t = \lceil \log n \rceil + 2$. For $i = 0, 1, \ldots, t$, let y_i be the number of columns of M with i ones. For $j = 1, \ldots, t$, let player j see all of M except row j. Then there exists an SM protocol in which each player sends $O(\log^2 n)$ bits to the referee, after which the referee can calculate $y_0, \ldots, y_t$.*

Proof. For $j = 1, \ldots, t$, player j sends $(a_j(0), a_j(1), \ldots, a_j(t-1))$ to the referee, where $a_j(i)$ is the number of columns player j sees with i ones. Note that each player sees only $t - 1$ of the rows and thus cannot see t ones in any column. For $i = 0, 1, \ldots, t-1$, the referee computes $b_i := \sum_{j=1}^{t} a_j(i)$.

We observe that $y_0, \ldots, y_t$ are nonnegative integers whose sum is $m \leq n$ and that for the b_i defined above they satisfy the system of equations (6.1). Thus, by Lemma 6.9, there is no other such solution. The referee, being arbitrarily powerful, can thus compute $y_0, \ldots, y_t$.

How many bits does each player send? Clearly, each $a_j(i) \leq n$, so each $a_j(i)$ can be communicated with $\lceil \log n \rceil$ bits. Since each player communicates $t = 1 + \lceil \log(n+1) \rceil$ such numbers to the referee, the complexity of this SM protocol is $O(\log^2 n)$ as desired. $\square$

We now state the theorem regarding SM protocols for SYMCOM functions.

THEOREM 6.11. *If (g, f) is a SYMCOM(n, k) problem and $k > 1 + \log n$, then $C_0(g \circ f) \leq \text{polylog}(n)$.*

Proof. Arrange the nk input bits of $g \circ f$ in a $k \times n$ matrix M such that player i knows all of M except the ith row. Each column of M contains the k input bits of a given f-gate. Let $t = \lceil \log n \rceil + 2$. The first t players will be the only ones who speak, so we call them the *active players*. We also call their rows and the entries in those rows *active*. The remaining players, rows, and entries are called *passive*.

Consider a single column $v \in \{0,1\}^k$ of M. Since f is compressible, there is a one-way 2-party protocol P for Alice and Bob, where Alice sees the passive entries of v and Bob sees the active entries of v, such that Alice sends Bob $O(\log t) = O(\log \log n)$ bits. Note that since f is symmetric, the only thing that Bob needs to know about the input he sees is how many ones there are. Thus, the value of $f(v)$ is determined by the message Alice sends on v and the number of ones among v's active entries.

For every column v of M, the t active players see all of the passive entries of v and thus know what message Alice would send under P upon seeing v. Let $c > 0$ be such that Alice sends at most $c \log \log n$ bits, and hence at most $r = \log^c n$ possible messages. Let $m_1, \ldots, m_r$ be the possible messages Alice can send under P.

From M, the active players form r new matrices $M_1, \ldots, M_r$, where M_i consists of the columns of M for which Alice sends Bob message m_i under protocol P. For each j, $1 \leq j \leq r$, the t players and the referee execute the protocol of Lemma 6.10

on the submatrix of M_j consisting of the active rows. Thus the referee can deduce, for each i, $0 \le i \le t$, the number of columns of M_j which have i ones among their active entries. From this and the fact that under P Alice would send message m_j for every column of M_j, the referee can deduce the number of columns v of M_j for which $f(v) = 1$. By summing over all j, the referee calculates the total number of f-gates that evaluate to 1, which suffices to evaluate $g \circ f$, as g is symmetric.

The cost of this protocol is $r \cdot O(\log^2 n) = O(\log^{c+2} n)$. □

6.1. A function which is not compressible. It is easy to check that a random symmetric function is incompressible. In this subsection we give an example of an explicit symmetric function which is not compressible (see Definition 6.1).

DEFINITION 6.12. *For an odd prime p, we define the function "quadratic character of the sum of the bits"* $\mathsf{QCSB}_p : \{0,1\}^p \to \{0,1\}$ *by* $\mathsf{QCSB}_p(x_1, \ldots, x_p) = 1$ *if and only if $x_1 + \cdots + x_p$ is a quadratic residue mod p, where the x_i are single bits. Recall that $y \neq 0$ is a quadratic residue mod p if y is a square mod p.*

Let p be an odd prime, and let $r = \lfloor (\frac{1}{2} - c_1) \log p \rfloor$, for any constant c_1, $0 < c_1 < \frac{1}{2}$. Let M be the $(r+1) \times (p+1)$ ± 1-matrix defined by $M(i,j) = 1$ if and only if $i + j$ is a quadratic residue mod p and -1 otherwise for $0 \le i \le r$, $0 \le j \le p$.

LEMMA 6.13. *For any $y \in \{-1,1\}^{r+1}$, the number of columns of M identical to y is $O(p/2^r)$.*

Proof. This is an immediate consequence of André Weil's character sum estimates (cf. [Sch76]; see also [Bo85, pp. 311, 319]): Let q be an odd prime power, and let U_0, U_1 be disjoint subsets of $\mathbb{F}_q$. For x, y in $\mathbb{F}_q$, let $\chi(x) = 1$ if $x \neq 0$ is a square in $\mathbb{F}_q$, 0 if $x = 0$, and -1 otherwise. Let

$$S = \{x \in \mathbb{F}_q \mid \text{for } i = 0, 1, (\forall u \in U_i)(\chi(x - u_i) = (-1)^i)\}.$$

Let $m = |U_0 \cup U_1|$, and let $s = |S|$. Then

$$|s - 2^{-m} q| \le m\sqrt{q}. \tag{6.4}$$

Let $q := p$. Let $y = (y_0, y_1, \ldots, y_r) \in \{-1,1\}^{r+1}$. Let $U_0 = \{p - i : 0 \le i \le r,\ y_i = 1\}$. Let $U_1 = \{p - i : 0 \le i \le r,\ y_i = -1\}$. Clearly, column j of M is identical to y exactly if $j \in S$. Setting $m := |U_0 \cup U_1| = r + 1$ gives us

$$s \le p/2^{r+1} + (r+1)\sqrt{p} = O(p/2^r). \qquad \square$$

THEOREM 6.14. *Let p and r be as above, and let b be an integer $r < b < (1 - c_2)p$, for any constant c_2. For any 2-party one-way (Alice to Bob) protocol for* QCSB_p*, if Bob sees b of the p input bits and Alice sees the other $a = p - b$ bits, then Alice must send Bob $r - O(1)$ bits.*

Proof. Assume that there is such a one-way protocol P. We set $b - r$ of Bob's bits to 0 and tell both players this. This is extra information, so P will still work for this restricted problem.

This new communication problem can be represented by the $(r+1) \times (a+1)$ ± 1-matrix M', obtained by deleting the last b columns of M. The rows represent the number of ones that Bob sees, and the columns represent the number of ones Alice sees. From Lemma 6.13, we know that every $y \in \{-1,1\}^{r+1}$ occurs in M and thus in M' at most $O(p/2^r)$ times. Since there are $p - b = \Omega(p)$ columns in M', the number of distinct columns in M' is $\Omega(2^r)$, and so Alice must send Bob $r - O(1)$ bits. □

COROLLARY 6.15. *If p is an odd prime, then* QCSB_p *is not compressible.*

Proof. If Bob sees $b = (\log p)^{O(1)}$ bits, then Alice must send at least $r - O(1) = \Omega(\log p) = b^{\Omega(1)}$ bits, which greatly exceeds the requirement for compressible functions. □

7. Decompositions of groups: The Rohrbach conjecture. In this section, we prove results about $\rho(G,u) := \rho_1(G,u)$ (i.e., when $\prod_{i=1}^{u} H_i = G$ in Definition 2.12) and indicate related conjectures. We shall see that $\rho(G,u)$ behaves roughly as $|G|^{1-1/u}$.

First, we observe the following easy lower bound.

PROPOSITION 7.1. *For any finite group G, $\rho(G,u) \geq |G|^{1-1/u}$.*

Proof. Let $G = H_1 \cdot \ldots \cdot H_u$ be an optimal decomposition. If $|H_i| \geq |G|^{1/u}$ for each i, then $|\widehat{H}_i| \geq |G|^{1-1/u}$. Otherwise, assume $|H_i| < |G|^{1/u}$ for some fixed i. Observe that $|H_i| \cdot |\widehat{H}_i| \geq |G|$. Therefore, in this case also $|\widehat{H}_i| > |G|^{1-1/u}$. □

For arbitrary finite groups, an upper bound on $\rho(G,u)$ that is not too far from optimal follows from Theorem 2.21.

COROLLARY 7.2 (of Theorem 2.21). *For any finite group G of order n, $\rho(G,u) \leq 2(4n\ln n)^{1-1/u}$.*

Proof. Partition the two-element sets of Theorem 2.21 into u classes, each containing at least $\lfloor m/u \rfloor$ of the m two-element sets. This means that there are at most $m - \lfloor m/u \rfloor$ two-element sets in any $u-1$ of the classes. Therefore,

$$\rho(G,u) \leq \max_i |\widehat{H}_i| \leq 2^{m-\lfloor m/u \rfloor} \leq 2^{1+m-m/u} = 2 \cdot 2^{m(u-1)/u} \leq 2(4n\ln n)^{1-1/u}. \qquad \square$$

The parameter $\rho(G,u)$ is closely related to a question posed by Rohrbach [Ro37a, Ro37b] in 1937.

DEFINITION 7.3. *A sequence $H_1, \ldots, H_u$ of subsets of a finite group G is called a* u-decomposition *of G if $G = H_1 \cdot \ldots \cdot H_u$. If $H_1 = \cdots = H_u = H$, we denote $H^u = H_1 \cdot \ldots \cdot H_u$. A subset H of G is called a* u-basis *of G if $H^u = G$.*

Rohrbach's Problem. Is there a constant $c = c(u)$ such that every finite group G has a u-basis H, where $|H| \leq c|G|^{1/u}$?

Note that if Rohrbach's Problem has an affirmative answer, then we will have, for every finite group G, $\rho(G,u) \leq (c(u))^{u-1}|G|^{1-1/u}$. On the other hand, if we have a u-decomposition $G = H_1 \cdot \ldots \cdot H_u$ with $|H_i| \leq \alpha|G|^{1/u}$, then we will have a u-basis H of G with $|H| \leq (\alpha u)|G|^{1/u}$ by simply taking $H = H_1 \cup \cdots \cup H_u$. Moreover, such a decomposition will also give that $\rho(G,u) \leq \alpha^{u-1}|G|^{1-1/u}$. Hence the following question appears to be of more general interest.

Modified Rohrbach Problem. Is there an absolute constant c such that every finite group G has a u-decomposition $G = H_1 \cdot \ldots \cdot H_u$, where $|H_i| \leq c|G|^{1/u}$?

Both Rohrbach's Problem and its modified version have been answered affirmatively for several special classes of finite groups. Of particular interest in our context is the following result of Kozma and Lev [KoL94] (for our purposes, $\lambda_i = 1/u$ in the following theorem).

THEOREM 7.4 (see [KoL94]). *Let G be a finite group such that every composition factor of G is either a cyclic group or an alternating group. Then for every positive integer u and nonnegative real numbers λ_i such that $\lambda_1 + \cdots + \lambda_u = 1$, there is a u-decomposition $G = H_1 \cdot \ldots \cdot H_u$, where $|H_1| \leq |G|^{\lambda_1}$ and $H_i \leq 2|G|^{\lambda_i}$ for $2 \leq i \leq u$. In particular, this conclusion holds if G is an alternating group or if G is solvable.*

This result answers the Modified Rohrbach Problem affirmatively for some important special classes of finite groups. As a consequence, we have the following corollary about $\rho(G,u)$.

COROLLARY 7.5. *Let G be a finite group such that every composition factor of G is either a cyclic group or an alternating group. Then for every positive integer u, $\rho(G,u) \le 2^{u-1}|G|^{1-1/u}$. In particular, this bound holds if G is an alternating group or if G is solvable.*

These results have been extended to groups with linear ($PSL(n,q)$) and symplectic ($Psp(2n,q)$) composition factors (in addition to cyclic and alternating composition factors) with a suitable small constant in place of the coefficient 2 (Pyber [Py98]). There is hope that the Modified Rohrbach Problem will be settled in the affirmative in the foreseeable future. (As mentioned above, this will also settle Rohrbach's original problem.)

In Examples 2.14 and 2.15, we gave upper bounds of $O(|G|^{1-1/u})$ on $\rho(G,u)$ of two special cases of greatest interest to us, namely $G = \mathbb{Z}_2^t$ and $G = \mathbb{Z}_n$. Thus in these cases we reduce the gap between the trivial lower bound $|G|^{1-1/u}$ (Proposition 7.1) and the upper bound to an absolute constant. These bounds are therefore stronger than those implied by an affirmative answer to the Modified Rohrbach Problem.

We propose the following stronger version of the Modified Rohrbach Problem.

Problem. Is there an absolute constant c such that for any finite group G and any positive integer u, $\rho(G,u) \le c|G|^{1-1/u}$?

We have given the positive answer for cyclic groups and for elementary abelian 2-groups.

8. Open problems.

1. The main open problem in multiparty communication complexity theory remains to find a nontrivial lower bound for some explicit function for more than $\log n$ players. The following are some candidate functions:
   (a) Let $\mathsf{T}_t^{k,r}$ be defined as follows: $\mathsf{T}_t^{k,r}(x_1,\dots,x_k) = 1$ if and only if $x_1 + \cdots + x_k \ge t$, where the x_i are r-bit integers. The candidate function is majority of thresholds $\mathsf{MT}_{n,k,r}$, defined by the following depth-2 circuit: the bottom level has n $\mathsf{T}_t^{k,r}$ gates, whose inputs are disjoint (so MT is a function of knr bits), and the top gate is a majority gate. There are k players, and the ith player misses the ith integer of each threshold gate. We recommend $n = k = r$.
   The (majority, $\mathsf{T}_t^{k,1}$)-communication problem is a SYMCOM(n,k) problem (see section 6), and hence for $k \ge 2 + \log n$ it has an efficient SM protocol. However, for $r \ge 2$, $\mathsf{MT}_{n,k,r}$ does not fit our description of SYMCOM functions (because players miss more than one input bit at each "f-gate"). So our protocol does not work, even though, for bounded r, $\mathsf{T}_t^{k,r}$ is a compressible function.
   (b) Quadratic character of the sum of the coordinates, defined as follows: Let p be an n-bit prime, and for $1 \le i \le k$ let x_i be an n-bit integer missed by player i. $\mathsf{QCS}_{p,k}(x_1,\dots,x_k) := 1$ if and only if $x_1+\cdots+x_k$ is a quadratic residue mod p. It is shown in [BaNS92] that $C(\mathsf{QCS}_{p,k}) \ge \Omega(n/2^k)$.
   (c) Let F be a finite field of order q. Give each player a $t \times t$ matrix over F. Let M be the product (in a given order) of these matrices. Estimate the SM complexity of decision problems associated with M, such as, "Is trace(M) a quadratic residue in F?" (for odd q). Here $n \approx q^{t^2}$. The case $t = 2$ is of particular interest.
2. Consider the SM problem $\mathsf{GAF}_{G,3}$. Show that there exists an $\epsilon > 0$ such that in any SM protocol for $\mathsf{GAF}_{G,3}$, if player 2 sends at most n^ϵ bits, then player 1 must send $\omega(n/\log\log n)$ bits. As explained in section 1.3,

such a lower bound would imply that the n-bit output function $f(x_0, x_1) = (\mathsf{GAF}_{G,3}(x_0, x_1, x_2))_{x_2 \in G}$ cannot be computed by circuits of size $O(n)$ and depth $O(\log n)$. Note that our lower bound proof in section 2 implies[3] that for *any* $\epsilon > 0$, if player 2 sends at most n^ϵ bits, then player 1 must send $\Omega(n^{1-\epsilon})$ bits. On the other hand, by the upper bound of [AmL00] (improving the protocol in section 5), it suffices if both players send $O(n^{0.73})$ bits.

3. Find nontrivial SM lower or upper bounds for the majority of QCSBs for any partition of the input bits (see section 6).
4. Obtain an $n^{1-\epsilon}$ SM upper bound for $\mathsf{GAF}_{G,k}$ for the case where G is a cyclic group. The best upper bound known is due to Ambainis [Am96]. He shows that $C_0(\mathsf{GAF}_{\mathbb{Z}_n,3}) = O(\frac{n \log^{1/4} n}{2^{\sqrt{\log n}}})$ and that $C_0(\mathsf{GAF}_{\mathbb{Z}_n,k}) = O(n^\epsilon)$ for an arbitrary $\epsilon > 0$ for $k = O((\log n)^{c(\epsilon)})$.
5. The lower bound on the randomized SM complexity of $\mathsf{GAF}_{\mathbb{Z}_2^t,k}$ from section 3 is useful only when the advantage ϵ is $n^{-O(1/k)}$. Improve this to yield meaningful (polylog(n)) lower bounds on the randomized SM complexity even for advantages as small as $2^{-\mathrm{polylog}(n)}$. Such an improvement would enable the extension of circuit lower bounds from section 4 to depth-3 in the spirit of [HG91, RaW93] exploiting an "approximation lemma" from [HMPST87]. Note that the [BaNS92] lower bound in the CFL model works even for advantages as small as $2^{-n/c^k}$.
6. *This problem, stated as an open question in an earlier version of this paper* [BaKL95]*, was resolved in* [BaHK01]*. We repeat the problem and state its current status.*
   Find a function f for which the one-way complexity $C_1(f)$ is exponentially larger than $C(f)$ for $k \geq 4$ players.
   Such a gap was found for $k = 3$ [NW93]; cf. Remark 2.9. For all $k \leq c \log n$, the gap was established in [BaHK01]. We note that lower bounds for C_1 complexity have been applied in [BaNS92] to lower bounds on branching programs and formula size.

Acknowledgments. We are grateful to Avi Wigderson for suggesting to us the SM model and pointing out the connection of ACC with the model considered in this paper. In October of 1994, P. Pudlák kindly sent us the manuscript [PRS97] by P. Pudlák, V. Rödl, and J. Sgall. Among many other things, that paper considers SM complexity under the name "oblivious communication complexity" and deduces the $\mathsf{GAF}_{G,k}$ lower bound for cyclic groups by essentially the same methods as ours [PRS97, Proposition 2.3].

Finally, we are grateful to an anonymous referee for suggestions which have lead to considerable improvement of our paper in two ways: the improvement discussed in Remark 2.10 and the use of entropy arguments for distributional complexity. The referee also pointed out to us the use of entropy in [Man98] and [KaT00].

REFERENCES

[Aj83] M. Ajtai, Σ_1^1-*formulae on finite structures*, Ann. Pure Appl. Logic, 24 (1983), pp. 1–48.

[3] In general, methods of section 2 can be used to prove that, in any SM protocol for $\mathsf{GAF}_{G,k}$, if player i sends ℓ_i bits, then $\prod_{i=1}^{k-1} \ell_i \geq n/2^{O(k \log k)}$ must hold.

[AlS00] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed., Wiley, New York, 2000.

[Am96] A. Ambainis, *Upper bounds on multiparty communication complexity of shifts*, in Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1046, Springer, Berlin, 1996, pp. 631–642.

[AmL00] A. Ambainis and S. V. Lokam, *Improved upper bounds on simultaneous messages complexity*, in Proceedings of LATIN 2000, Lecture Notes in Comput. Sci. 1776, Springer, Berlin, 2000, pp. 207–216.

[BaE82] L. Babai and P. Erdős, *Representation of group elements as short products*, Ann. Discrete Math., 12 (1982), pp. 21–26.

[BaHK01] L. Babai, T. Hayes, and P. Kimmel, *The cost of the missing bit: Communication complexity with help*, Combinatorica, 21 (2001), pp. 455–488.

[BaKL95] L. Babai, P. Kimmel, and S. V. Lokam, *Simultaneous messages vs. communication*, in Proceeedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 900, Springer, Berlin, 1995, pp. 361–372.

[BaNS92] L. Babai, N. Nisan, and M. Szegedy, *Multiparty protocols, pseudorandom generators for logspace and time-space trade-offs*, J. Comput. System Sci., 45 (1992), pp. 204–232.

[BeT91] R. Beigel and J. Tarui, *On ACC*, in Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, San Juan, PR, 1991, pp. 783–792.

[BJKS02] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar, *Information theory methods in communication complexity*, in Proceedings of the 17th IEEE Conference on Computational Complexity, Montreal, QB, Canada, 2002, pp. 72–81.

[Bo85] B. Bollobás, *Random Graphs*, Academic Press, London, 1985.

[CFL83] A. K. Chandra, M. L. Furst, and R. J. Lipton, *Multiparty protocols*, in Proceedings of the 15th Annual ACM Symposium on Theory of Computing, Boston, MA, 1983, pp. 94–99.

[CT91] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991.

[EGS75] P. Erdős, R. Graham, and E. Szemerédi, *On sparse graphs with dense long paths*, Comp. and Maths. with Appls., 1 (1975), pp. 365–369.

[FSS84] M. Furst, J. Saxe, and M. Sipser, *Parity, circuits, and the polynomial time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.

[G94] V. Grolmusz, *The BNS lower bound for multi-party protocols is nearly optimal*, Inform. and Comput., 112 (1994), pp. 51–54.

[HG91] J. Håstad and M. Goldmann, *On the power of small-depth threshold circuits*, Comput. Complexity, 1 (1991), pp. 113–129.

[HMPST87] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán, *Threshold circuits of bounded depth*, in Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, Los Angeles, CA, 1987, pp. 99–110.

[KaT00] J. Katz and L. Trevisan, *On the efficiency of local decoding procedures for error-correcting codes*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 80–86.

[KoL94] G. Kozma and A. Lev, *On H-bases and H-decompositions of finite solvable and alternating groups*, J. Number Theory, 49 (1994), pp. 385–391.

[KuN97] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.

[MacS77] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Elsevier, Amsterdam, 1977.

[Man98] E. Mann, *Private Access to Distributed Information*, Master's thesis, Technion, Haifa, Israel, 1998.

[MNT93] Y. Mansour, N. Nisan, and P. Tiwari, *The computational complexity of universal hashing*, Theoret. Comput. Sci., 107 (1993), pp. 121–133.

[NW93] N. Nisan and A. Wigderson, *Rounds in communication complexity revisited*, SIAM J. Comput., 22 (1993), pp. 211–219.

[PR93] P. Pudlák and V. Rödl, *Modified ranks of tensors and the size of circuits*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 523–531.

[PRS97] P. Pudlák, V. Rödl, and J. Sgall, *Boolean circuits, tensor ranks, and communication complexity*, SIAM J. Comput., 26 (1997), pp. 605–633.

[Py98] L. Pyber, *On the Rohrbach Conjecture*, private communication.

[RaW93] A. Razborov and A. Wigderson, $n^{\Omega(\log n)}$ *lower bounds on the size of depth* 3

circuits with AND gates at the bottom, Inform. Process. Lett., 45 (1993), pp. 303–307.

[Ro37a] H. Rohrbach, *Ein Beitrag zur additiven Zahlentheorie*, Math. Z., 42 (1937), pp. 1–30.

[Ro37b] H. Rohrbach, *Anwendung eines Satzes der additiven Zahlentheorie auf eine gruppentheoretische Frage*, Math. Z., 42 (1937), pp. 538–542.

[Sch76] W. M. Schmidt, *Equations over Finite Fields: An Elementary Approach*, Lecture Notes in Math. 536, Springer, Berlin, 1976.

[Va77] L. Valiant, *Graph-theoretic arguments in low-level complexity*, in Proceedings of 6th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 53, Springer, Berlin, 1977, pp. 162–176.

[Ya83] A. C.-C. Yao, *Lower bounds by probabilistic arguments*, in Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science, Tucson, AZ, 1983, pp. 420–428.

[Ya90] A. C.-C. Yao, *On ACC and threshold circuits*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 619–627.

SIAM J. COMPUT.
Vol. 33, No. 1, pp. 167–226

DESIGN AND ANALYSIS OF PRACTICAL PUBLIC-KEY ENCRYPTION SCHEMES SECURE AGAINST ADAPTIVE CHOSEN CIPHERTEXT ATTACK*

RONALD CRAMER† AND VICTOR SHOUP‡

Abstract. A new public-key encryption scheme, along with several variants, is proposed and analyzed. The scheme and its variants are quite practical and are proved secure against adaptive chosen ciphertext attack under standard intractability assumptions. These appear to be the first public-key encryption schemes in the literature that are simultaneously practical and provably secure.

Key words. cryptography, public-key encryption, chosen ciphertext security, decisional Diffie–Hellman assumption

AMS subject classifications. 94A60, 68P25

DOI. 10.1137/S0097539702403773

1. Introduction.

In this paper, we present and analyze a new public-key encryption scheme, and several variants, proving that they are secure against adaptive chosen ciphertext attack (as defined by Rackoff and Simon [50]) under standard intractability assumptions. The schemes are quite practical, requiring just a few exponentiations in a group for both encryption and decryption. Moreover, the proofs of security of these schemes rely only on standard intractability assumptions: one variant relies only on the hardness of the decisional Diffie–Hellman problem, while other, somewhat more practical, variants rely on a couple of other standard intractability assumptions.

The hardness of the decisional Diffie–Hellman problem is essentially equivalent to the semantic security of the basic ElGamal encryption scheme [30]. Thus, with just a bit more computation, we get security against adaptive chosen ciphertext attack, whereas the basic ElGamal scheme is completely insecure against this type of attack.

While there are several provably secure public-key encryption schemes in the literature, they are all quite impractical. Also, there are several practical encryption schemes that have been proposed, but none of them has been proven secure under standard intractability assumptions. The significance of our results is that they provide several schemes that are provably secure and practical at the same time. There appear to be no other public-key encryption schemes in the literature that enjoy both of these properties simultaneously.

1.1. Chosen ciphertext security.

Semantic security, defined by Goldwasser and Micali [34], captures the intuition that an adversary should not be able to obtain any partial information about a message given its encryption. However, this guarantee of secrecy is only valid when the adversary is completely passive, i.e., can only eavesdrop. Indeed, semantic security offers no guarantee of secrecy at all if an adversary

*Received by the editors March 11, 2002; accepted for publication (in revised form) August 31, 2003; published electronically December 31, 2003. This paper is a significantly revised and extended version of the extended abstract [22] and also includes results originally presented in the extended abstract [56].

http://www.siam.org/journals/sicomp/33-1/40377.html

†BRICS, Department of Computer Science, Aarhus University, Ny Munkegade, 8000 Aarhus C., Denmark (cramer@brics.dk).

‡Department of Computer Science, New York University, 251 Mercer Street, New York, NY 10012 (shoup@cs.nyu.edu).

can mount an active attack, i.e., inject messages into a network or otherwise influence the behavior of parties in the network.

To deal with active attacks, Rackoff and Simon [50] defined the notion of security against an *adaptive chosen ciphertext attack*. If an adversary can inject messages into a network, these messages may be ciphertexts, and the adversary may be able to extract partial information about the corresponding cleartexts through its interactions with the parties in the network. Rackoff and Simon's definition models this type of attack by simply allowing an adversary to obtain decryptions of its choice; i.e., the adversary has access to a "decryption oracle." Now, given an encryption of a message—the "target" ciphertext—we want to guarantee that the adversary cannot obtain any partial information about the message. To achieve this, we have to restrict the adversary's behavior in some way; otherwise, the adversary could simply submit the target ciphertext itself to the decryption oracle. The restriction proposed by Rackoff and Simon is the weakest possible: the adversary is not allowed to submit the target ciphertext itself to the oracle; however, it may submit any other ciphertext, including ciphertexts that are related to the target ciphertext.

A different notion of security against active attacks, called *nonmalleability*, was proposed by Dolev, Dwork, and Naor [27, 28]. Here, the adversary also has access to a decryption oracle, but his goal is not to obtain partial information about the target ciphertext but rather to create another encryption of a different message that is related in some interesting way to the original, encrypted message. For example, for a nonmalleable encryption scheme, given an encryption of n, it should be infeasible to create an encryption of $n + 1$. It turns out that nonmalleability and security against adaptive chosen ciphertext attack are equivalent [8, 28].

An encryption scheme secure against adaptive chosen ciphertext attack is a very powerful cryptographic primitive. It is essential in designing protocols that are secure against active adversaries. For example, this primitive is used in protocols for authentication and key exchange [29, 28, 54] and in protocols for escrow, certified e-mail, and more general fair exchange [3]. It is by now generally recognized in the cryptographic research community that security against adaptive chosen ciphertext attack is the "right" notion of security for a general-purpose public-key encryption scheme. This is exemplified by the adoption of Bellare and Rogaway's OAEP scheme [10] (a practical but only heuristically secure scheme) as the Internet encryption standard RSA PKCS#1 version 2 and for use in the SET protocol for electronic commerce. Another motivation for security against adaptive chosen ciphertext attack is Bleichenbacher's attack [13] on the widely used SSL key establishment protocol, which is based on RSA PKCS#1 version 1—Bleichenbacher showed how to break this protocol by mounting a specific chosen ciphertext attack (SSL still uses RSA PKCS#1 version 1, but the protocol has been patched so as to avoid Bleichenbacher's attack).

There are also intermediate notions of security between semantic security and adaptive chosen ciphertext security. Naor and Yung [47] propose an attack model where the adversary has access to the decryption oracle only *prior* to obtaining the target ciphertext, and the goal of the adversary is to obtain partial information about the encrypted message. Naor and Yung called this type of attack a *chosen ciphertext attack*; it has also been called a "lunch-time" or "midnight" attack, and also an *indifferent* chosen ciphertext attack. In this paper, we will use the phrase *adaptive* chosen ciphertext attack for Rackoff and Simon's definition to distinguish it from Naor and Yung's definition. Also, throughout this paper, unless otherwise specified, by "security" we will always mean "security against adaptive chosen ciphertext attack."

1.2. Previous work.

Provably secure schemes. Naor and Yung [47] presented the first scheme provably secure against lunch-time attacks. Subsequently, Dolev, Dwork, and Naor [27] presented a scheme that is provably secure against adaptive chosen ciphertext attack.

Rackoff and Simon [50] present and prove the security of an encryption scheme, but their scheme is actually not a public-key scheme in the traditional sense: in their scheme, *all users*—both senders and receivers—require public keys, and, moreover, a trusted center is required to perform certain functions. In contrast, all other schemes mentioned in this paper, including our own, are traditional public-key systems: encryption is a probabilistic function of the message and the receiver's public key, decryption is a function of the ciphertext and the receiver's secret key, and no trusted center is required. This distinction can be important: adding extra system requirements as in the Rackoff and Simon scheme can greatly restrict the range of application of the scheme.

All of the previously known schemes provably secure under standard intractability assumptions are completely impractical (albeit polynomial time), as they rely on general and expensive constructions for noninteractive zero-knowledge proofs. This includes nonstandard schemes like Rackoff and Simon's as well.

Practical schemes. Damgård [25] proposed a practical scheme that he conjectured to be secure against lunch-time attacks; however, this scheme is not known to be provably secure in this sense and is in fact demonstrably insecure against adaptive chosen ciphertext attack.

Zheng and Seberry [62] proposed practical schemes that are conjectured to be secure against chosen ciphertext attack, but again no proof based on standard intractability assumptions is known. Lim and Lee [39] also proposed practical schemes that were later broken by Frankel and Yung [31].

Bellare and Rogaway [9, 10] have presented practical schemes for which they give heuristic proofs of adaptive chosen ciphertext security; namely, they prove security based on the assumption of a one-way trapdoor permutation in an idealized model of computation, the so-called *random oracle* model, wherein a hash function is represented by a random oracle. Actually, it turns out that the proof of security of the OAEP scheme in [10] is flawed: as demonstrated in [57], there can be no standard "black box" security proof based on an arbitrary one-way trapdoor permutation. However, the negative result in [57] does not rule out the possibility that OAEP in conjunction with a specific one-way trapdoor permutation scheme is secure. Indeed, it is shown in [57] that OAEP with exponent-3 RSA is secure, and this result is generalized in [32] to arbitrary-exponent RSA. A new scheme, OAEP+, is also presented in [57], which can be proven secure in the random oracle model, using an arbitrary one-way trapdoor permutation. Further variations of OAEP and OAEP+ are discussed in [15].

Shoup and Gennaro [58] also give ElGamal-like schemes that are secure against adaptive chosen ciphertext attack in the random oracle model and that are also amenable to efficient threshold decryption.

We stress that although a security proof in the random oracle model is of some value, it is still only a heuristic proof. In particular, these types of proofs do not rule out the possibility of breaking the scheme without breaking the underlying intractability assumption. Nor do they even rule out the possibility of breaking the scheme without finding some kind of weakness in the hash function, as shown by Canetti, Goldreich, and Halevi [19].

1.3. Further progress. Subsequent to the publication of the extended abstract [22] on which the present paper is based, some further progress in this area has been made. Canetti and Goldwasser [20] presented a threshold-decryption variant of our scheme. Also, the authors of the present paper [24] have generalized and extended the basic ideas underlying our encryption scheme, obtaining new and quite practical encryption schemes that are secure against adaptive chosen ciphertext attack under different assumptions—one scheme relies on Paillier's decision composite residuosity assumption [49], while the other (somewhat less practical) scheme relies on the classical quadratic residuosity assumption.

1.4. Outline of paper. Our paper consists of two parts.

Part 1. In the first part, we take care of a number of preliminaries, after which we present a basic version of our new scheme, along with a few variants. This first part is organized as follows:

Section 2: We introduce some basic notation that will be used throughout the paper.

Section 3: We state the formal definition of a public-key encryption scheme and the notion of security against adaptive chosen ciphertext attack. We also discuss some implications of the definition of security that illustrate its utility.

Section 4: We state the formal definitions of several intractability assumptions related to the discrete logarithm problem: the discrete logarithm assumption, the computational Diffie–Hellman assumption, and the decisional Diffie–Hellman assumption. In doing this, we introduce the notion of a *computational group scheme*, which is a general framework that allows us to discuss in an abstract, yet sufficiently concrete way the different families of groups that may be used in cryptographic applications.

Section 5: We define the notion of a *target collision resistant hash function*, which is a special type of a *universal one-way hash function*. We will use this primitive in the most efficient variants of our encryption scheme.

Section 6: We present and analyze the basic version of our encryption scheme, which we call CS1, along with two variants, called CS1a and CS1b. We prove the security of these schemes based on the decisional Diffie–Hellman assumption and the assumption that a given family of hash functions is target collision resistant. We also present and analyze a somewhat less efficient scheme, called CS2, which does not require a target collision resistant hash function.

Part 2. The schemes presented in section 6 suffer from two drawbacks. First, the schemes require that plaintexts are, or can be encoded as, group elements, which may significantly restrict the range of application of the encryption scheme and/or the choice of computational group scheme; it would be nice to relax this restriction, allowing plaintexts to be, say, bit strings of arbitrary length. Second, if the decisional Diffie–Hellman assumption is false, these schemes can be trivially broken; it would be nice if we could provide a second level of defense so that if the decisional Diffie–Hellman assumption turns out to be false, we have a scheme that still offers some security—even if only heuristically.

It turns out that both of these drawbacks can be dealt with by using a technique called *hybrid encryption*. Basically, a hybrid encryption scheme uses public-key encryption techniques to derive a shared key that is then used to encrypt the actual message using standard symmetric-key techniques. The second part of the paper is devoted to developing the formal theory behind this technique and to designing and analyzing variations on our basic scheme that utilize this technique. This part is organized as follows:

Section 7: We lay the theoretical foundations for hybrid encryption. Although most of the ideas in this section appear to be "folklore," they have not been treated rigorously in the literature. In section 7.1, we introduce the notion of a *key encapsulation mechanism* and an appropriate notion of security against adaptive chosen ciphertext attack. A key encapsulation mechanism is like a public-key encryption scheme, except that the encryption algorithm can only be used to generate and encrypt a random bit string of fixed length, which we shall use as a key for a symmetric-key encryption scheme. In section 7.2, we state the formal properties of a symmetric-key encryption scheme that we need for use in a hybrid encryption scheme and discuss some simple constructions based on pseudorandom bit generators and message authentication codes. In section 7.3, we prove that an appropriately secure key encapsulation mechanism, combined with an appropriately secure symmetric-key encryption scheme, yields a public-key encryption scheme that is secure against adaptive chosen ciphertext attack.

In what follows, we concentrate exclusively on the problem of constructing secure key encapsulation mechanisms, since the problem of constructing symmetric-key encryption schemes is essentially solved.

Section 8: We discuss the notion of a secure *key derivation function*, which is a function that should map random group elements to pseudorandom bit strings of a given length. A key derivation function is an essential ingredient in our constructions of key encapsulation mechanisms.

Section 9: We present and analyze a key encapsulation mechanism, CS3, along with two variants, CS3a and CS3b, and prove their security under the decisional Diffie–Hellman assumption, and also assuming a target collision resistant hash function and a secure key derivation function.

Section 10: The hybrid encryption scheme obtained from CS3b is by far the most practical of the encryption schemes presented in this paper; moreover, it has other interesting security properties. We show that CS3b is no less secure than a more traditional key encapsulation mechanism that is a hashed variant of ElGamal encryption, which we call HEG. Second, we also show that CS3b is secure in the random oracle model (viewing the key derivation function as a random function), under the weaker *computational* Diffie–Hellman assumption, and also assuming a target collision resistant hash function. The results in this section show that there is virtually no risk in using scheme CS3b relative to more traditional encryption schemes, while at the same time CS3b provides a security guarantee that more traditional schemes do not.

We conclude in section 11 with a brief summary.

2. Some preliminaries.

2.1. Basic mathematical notation. $\mathbf{Z}$ denotes the ring of integers, $\mathbf{Z}_{\geq 0}$ denotes the set of nonnegative integers, and for positive integer k, $\mathbf{Z}_k$ denotes the ring of integers modulo k, and $\mathbf{Z}_k^*$ denotes the corresponding multiplicative group of units.

2.2. Algorithms and probability spaces. We write $\nu \leftarrow \alpha$ to denote the algorithmic action of assigning the value of α to the variable ν.

Let X be a finite probability space, i.e., a probability space on a finite set S. For $\alpha \in S$, we let $\Pr_X[\alpha]$ denote the probability that X assigns to α, and for $S' \subset S$ we let $\Pr_X[S']$ denote the probability that X assigns to S'.

We write $\nu \xleftarrow{R} X$ to denote the algorithmic action of sampling an element of S

according to the distribution X and assigning the result of this sampling experiment to the variable ν. We sometimes write $\nu_1, \ldots, \nu_k \xleftarrow{R} X$ as shorthand for $\nu_1 \xleftarrow{R} X; \ldots; \nu_k \xleftarrow{R} X$.

For any finite set S, $\mathbf{U}(S)$ denotes the uniform distribution on S. We write $\nu \xleftarrow{R} S$ as shorthand for $\nu \xleftarrow{R} \mathbf{U}(S)$.

For any probability space X on a finite set S, we denote by $[X]$ the subset of elements of S that are assigned nonzero probability by X, i.e., the "support" of X.

If $X_1, X_2, \ldots, X_k$ are finite probability spaces and ϕ is a k-ary predicate, then we write

$$\Pr[\phi(\nu_1, \ldots, \nu_k) : \nu_1 \xleftarrow{R} X_1; \ldots; \nu_k \xleftarrow{R} X_k]$$

to denote the probability that $\phi(\nu_1, \ldots, \nu_k)$ holds when ν_1 is sampled from X_1, ν_2 is sampled from X_2, etc. More generally, for $1 \leq i \leq k$, X_i may be a family of finite probability spaces parameterized by $(\nu_1, \ldots, \nu_{i-1})$, and we write

$$\Pr[\phi(\nu_1, \ldots, \nu_k) : \nu_1 \xleftarrow{R} X_1; \nu_2 \xleftarrow{R} X_2(\nu_1); \ldots; \nu_k \xleftarrow{R} X_k(\nu_1, \ldots, \nu_{k-1})]$$

to denote the probability that $\phi(\nu_1, \ldots, \nu_k)$ holds when ν_1 is sampled from X_1, after which ν_2 is sampled from $X_2(\nu_1)$, and so on. In this case, it is important that $\nu_1, \ldots, \nu_k$ are sampled in the order given.

Similarly, if F is a k-ary function, then

$$\{F(\nu_1, \ldots, \nu_k) : \nu_1 \xleftarrow{R} X_1; \nu_2 \xleftarrow{R} X_2(\nu_1); \ldots; \nu_k \xleftarrow{R} X_k(\nu_1, \ldots, \nu_{k-1})\}$$

denotes the probability space defined by sampling ν_1 from X_1, ν_2 from $X_2(\nu_1)$, and so on, and then evaluating the function $F(\nu_1, \ldots, \nu_k)$.

We shall consider polynomial-time probabilistic algorithms A. We shall insist that for all $\lambda \in \mathbf{Z}_{\geq 0}$ and all inputs of length λ, algorithm A *always* halts in time bounded by a polynomial in λ, regardless of the random choices that A may make. In particular, for any input tuple $(\alpha_1, \ldots, \alpha_k)$, the random choices made by A as well as the output of A on this input are finite probability spaces. We denote this output probability space of A for a given input $(\alpha_1, \ldots, \alpha_k)$ by $\mathsf{A}(\alpha_1, \ldots, \alpha_k)$. We stress that $\mathsf{A}(\alpha_1, \ldots, \alpha_k)$ is a *probability space* and not a *value*. As such, we may write $\nu \xleftarrow{R} \mathsf{A}(\alpha_1, \ldots, \alpha_k)$ to denote the algorithmic action of running A on input $(\alpha_1, \ldots, \alpha_k)$ and assigning the output to the variable ν. When we speak of the "running time" of A, we mean the worst-case running time of A for inputs of a given length.

To exercise the above notation a bit, note that $[\mathsf{A}(\alpha_1, \ldots, \alpha_k)]$ denotes the set of possible outputs of A on input $(\alpha_1, \ldots, \alpha_k)$. For a tertiary predicate ϕ, polynomial-time probabilistic algorithms A_1 and A_2, and a value α_0,

$$\Pr[\phi(\alpha_0, \alpha_1, \alpha_2) : \alpha_1 \xleftarrow{R} \mathsf{A}_1(\alpha_0); \alpha_2 \xleftarrow{R} \mathsf{A}_2(\alpha_0, \alpha_1)]$$

denotes the probability that $\phi(\alpha_0, \alpha_1, \alpha_2)$ holds when A_1 is run on input α_0, yielding an output α_1, and then A_2 is run on input (α_0, α_1), yielding an output α_2.

For $\lambda \in \mathbf{Z}_{\geq 0}$, $\mathtt{1}^\lambda$ denotes the bit string consisting of λ copies of the bit $\mathtt{1}$. The string $\mathtt{1}^\lambda$ will often be an input to an algorithm: this is a technical device that allows a polynomial-time algorithm to run in time bounded by a polynomial in λ, even if there are no other inputs to the algorithm or those inputs happen to be very short.

2.3. Statistical distance and negligible functions. Let X and Y be probability spaces on a finite set S. Define the *statistical distance* $\Delta(X,Y)$ *between* X *and* Y as

$$\Delta(X,Y) := \frac{1}{2}\sum_{\alpha\in S} |\Pr_X[\alpha] - \Pr_Y[\alpha]|.$$

One can easily verify that

$$\Delta(X,Y) = \max_{S'\subset S} |\Pr_X[S'] - \Pr_Y[S']|.$$

A function F mapping nonnegative integers to nonnegative reals is called *negligible* if for all positive numbers c there exists an integer $\lambda_0(c) \geq 0$ such that for all $\lambda > \lambda_0(c)$ we have $F(\lambda) < 1/\lambda^c$.

3. Secure public-key encryption. In this section, we state the basic properties of a public-key encryption scheme, along with the definition of security against adaptive chosen ciphertext attack. Although the notions here are relatively standard, we treat a number of details here that are not often dealt with in the literature. We also discuss some implications of the definition of security that illustrate its utility.

3.1. Public-key encryption schemes. A *public-key encryption scheme* PKE consists of the following algorithms:

- A probabilistic, polynomial-time *key generation algorithm* $\mathsf{PKE.KeyGen}$ that on input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$ outputs a public-key/secret-key pair $(\mathsf{PK}, \mathsf{SK})$. The structure of PK and SK depends on the particular scheme.
  For $\lambda \in \mathbf{Z}_{\geq 0}$, we define the probability spaces

  $$\mathsf{PKE.PKSpace}_\lambda := \{\mathsf{PK} : (\mathsf{PK},\mathsf{SK}) \xleftarrow{R} \mathsf{PKE.KeyGen}(1^\lambda)\}$$

  and

  $$\mathsf{PKE.SKSpace}_\lambda := \{\mathsf{SK} : (\mathsf{PK},\mathsf{SK}) \xleftarrow{R} \mathsf{PKE.KeyGen}(1^\lambda)\}.$$

- A probabilistic, polynomial-time encryption algorithm $\mathsf{PKE.Encrypt}$ that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a public key $\mathsf{PK} \in [\mathsf{PKE.PKSpace}_\lambda]$, and a message m and outputs a ciphertext ψ.
  A ciphertext is a bit string. The structure of a message may depend on the particular scheme; see below (section 3.1.1) for a discussion.
- A deterministic, polynomial-time decryption algorithm $\mathsf{PKE.Decrypt}$ that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a secret key $\mathsf{SK} \in [\mathsf{PKE.SKSpace}_\lambda]$, and a ciphertext ψ and outputs either a message m or the special symbol reject.

3.1.1. Message spaces. Different public-key encryption schemes might specify different message spaces, and these message spaces might in fact vary with the choice of public key. Let us denote by $\mathsf{PKE.MSpace}_{\lambda,\mathsf{PK}}$ the message space associated with $\lambda \in \mathbf{Z}_{\geq 0}$ and $\mathsf{PK} \in [\mathsf{PKE.PKSpace}_\lambda]$. Although there may be other ways of categorizing message spaces, we shall work with schemes that specify message spaces in one of two ways:

unrestricted message space: $\mathsf{PKE.MSpace}_{\lambda,\mathsf{PK}} = \{0,1\}^*$ for all λ and PK.

restricted message space: $\mathsf{PKE.MSpace}_{\lambda,\mathsf{PK}}$ is a finite set that may depend on λ and PK.

There should be a deterministic, polynomial-time algorithm that on input 1^λ, PK, and α determines if $\alpha \in \mathsf{PKE.MSpace}_{\lambda,\mathsf{PK}}$.

Clearly, a public-key encryption scheme with an unrestricted message space will be most suitable in a setting where a very general-purpose encryption scheme is required. However, encryption schemes with restricted message spaces can be useful in some settings as well.

3.1.2. Soundness. A public-key encryption scheme should be *sound* in the sense that decrypting an encryption of a message should yield the original message.

Requiring that this *always* holds is a very strong condition which will not be satisfied by many otherwise quite acceptable encryption schemes.

A definition of soundness that is adequate for our purposes runs as follows. Let us say a public-key/secret-key pair $(\mathsf{PK}, \mathsf{SK}) \in [\mathsf{PKE.KeyGen}(1^\lambda)]$ is *bad* if for some $m \in \mathsf{PKE.MSpace}_{\lambda,\mathsf{PK}}$ and some $\psi \in [\mathsf{PKE.Encrypt}(1^\lambda, \mathsf{PK}, m)]$ we have $\mathsf{PKE.Decrypt}(1^\lambda, \mathsf{SK}, \psi) \neq m$. Then our requirement is that the probability that the key generation algorithm outputs a bad key pair grows negligibly in λ.

One could formulate even weaker notions of soundness that would still be adequate for many applications, but we shall not pursue them here.

3.2. Security against adaptive chosen ciphertext attack. An adversary A in an adaptive chosen ciphertext attack (CCA) is a probabilistic, polynomial-time *oracle query* machine.

The attack game is defined in terms of an interactive computation between the adversary and its *environment*. The adversary's environment responds to the oracle queries made by the adversary: each oracle query response is sampled from a probability space that is a function of the adversary's input and all the previous oracle queries made by the adversary. We require that A runs in time *strictly* bounded by a polynomial in the length of its input, regardless of its probabilistic choices and regardless of the responses to its oracle queries from its environment.

We now describe the attack game used to define security against adaptive chosen ciphertext attack; that is, we define (operationally) the environment in which A runs. We assume that the input to A is 1^λ for some $\lambda \in \mathbf{Z}_{\geq 0}$.

Stage 1: The adversary queries a *key generation oracle*. The key generation oracle computes $(\mathsf{PK}, \mathsf{SK}) \xleftarrow{R} \mathsf{PKE.KeyGen}(1^\lambda)$ and responds with PK.

Stage 2: The adversary makes a sequence of calls to a *decryption oracle*.

For each decryption oracle query, the adversary submits a bit string ψ, and the decryption oracle responds with $\mathsf{PKE.Decrypt}(1^\lambda, \mathsf{SK}, \psi)$.

We emphasize that ψ may be an arbitrary bit string, concocted by A in an arbitrary fashion—it certainly need not be an output of the encryption algorithm.

Stage 3: The adversary submits two messages $m_0, m_1 \in \mathsf{PKE.MSpace}_{\lambda,\mathsf{PK}}$ to an *encryption oracle*. In the case of an unrestricted message space, we require that $|m_0| = |m_1|$.

On input m_0, m_1, the encryption oracle computes

$$\sigma \xleftarrow{R} \{0,1\};\ \psi^* \xleftarrow{R} \mathsf{PKE.Encrypt}(1^\lambda, \mathsf{PK}, m_\sigma)$$

and responds with the "target" ciphertext ψ^*.

Stage 4: The adversary continues to make calls to the decryption oracle subject only to the restriction that a submitted bit string ψ is not *identical* to ψ^*.

Again, we emphasize that ψ is arbitrary and may even be computed by A as a function of ψ^*.

Stage 5: The adversary outputs $\hat{\sigma} \in \{0,1\}$.

We define the *CCA advantage of* A *against* PKE *at* λ, denoted $\mathsf{AdvCCA}_{\mathsf{PKE},\mathsf{A}}(\lambda)$, to be $|\Pr[\sigma = \hat{\sigma}] - 1/2|$ in the above attack game.

We say that PKE is *secure against adaptive chosen ciphertext attack* if

> *for all probabilistic, polynomial-time oracle query machines* A *the function* $\mathsf{AdvCCA}_{\mathsf{PKE},\mathsf{A}}(\lambda)$ *grows negligibly in* λ.

3.3. Alternative characterization of security. In applying the above definition of security, one typically works directly with the quantity

$$\mathsf{AdvCCA}'_{\mathsf{PKE},\mathsf{A}}(\lambda) := |\Pr[\hat{\sigma} = 1 \mid \sigma = 0] - \Pr[\hat{\sigma} = 1 \mid \sigma = 1]|\,.$$

If we view A as a statistical test, then the quantity $\mathsf{AdvCCA}'_{\mathsf{PKE},\mathsf{A}}(\lambda)$ measures A's advantage in distinguishing a game in which m_0 is always encrypted from a game in which m_1 is always encrypted. It is easy to verify that

$$\mathsf{AdvCCA}'_{\mathsf{PKE},\mathsf{A}}(\lambda) = 2 \cdot \mathsf{AdvCCA}_{\mathsf{PKE},\mathsf{A}}(\lambda).$$

We present here a sketch of another characterization of this notion of security that illustrates more fully its utility in reasoning about the security of higher-level protocols. This alternative characterization is a natural, high-level, simulation-based definition that in some ways provides a justification for the rather low-level, technical definition given above. Our treatment here will be somewhat less formal than elsewhere in this paper.

We start by defining the notion of a *channel*, which is an object that implements the following operations:

- KeyGen—outputs a public key PK.
- Encrypt—takes as input a message m and outputs a ciphertext ψ.
- Decrypt—takes as input a ciphertext ψ and outputs a message m (possibly a special reject symbol).

Additionally, a channel is parameterized by a security parameter λ.

To initialize a channel, the KeyGen operation is invoked, after which an arbitrary number of Encrypt and Decrypt operations may be invoked. A channel may maintain state between invocations of these operations. We shall assume that messages are arbitrary bit strings.

A channel may be implemented in several ways. One way, of course, is to simply "plug in" a public-key encryption scheme. We call such an implementation of a channel a *real channel*. We wish to describe another implementation, which we call an *ideal channel*.

Loosely speaking, an ideal channel acts essentially like a private storage and retrieval service: when an Encrypt operation is invoked with a message m, the ideal channel generates a corresponding ciphertext ψ without even "looking" at m and stores the pair (m, ψ) in a table; when a Decrypt operation is invoked with a ciphertext ψ such that (m, ψ) is in the table for some m, the ideal channel returns the message m. Thus, the "encryption" ψ of a message m is completely independent of m and essentially plays the role of a "receipt," which when presented to the Decrypt operation yields the message m. As such, the Encrypt operation might be better named Store and the Decrypt operation named Retrieve.

We now describe the operation of an ideal channel in a bit more detail.

An ideal channel is built using a *channel simulator*. A channel simulator is an object that implements an interface that is identical to that of a channel, except that

the Encrypt operation does not take as input a message but rather just the *length* of a message.

An ideal channel uses a channel simulator as follows. The KeyGen operation of the ideal channel is implemented directly in terms of the KeyGen operation of the channel simulator. The ideal channel maintains a set S of message/ciphertext pairs (m, ψ) and a set T of ciphertexts, both initially empty.

When the Encrypt operation of the ideal channel is invoked with input m, the ideal channel invokes the channel simulator with input $|m|$, obtaining a ciphertext ψ. If $\psi \in T$ or $(m', \psi) \in S$ for some m', the ideal channel becomes "broken," and this and all subsequent invocations of either Encrypt or Decrypt return a special symbol indicating this; otherwise, the ideal channel adds the pair (m, ψ) to S and returns ψ as the result of the Encrypt operation.

When the Decrypt operation of the ideal channel is invoked with input ψ, the ideal channel first checks if $(m, \psi) \in S$ for some m; if so, it simply returns the message m; otherwise, it adds ψ to T, invokes the Decrypt operation of the channel simulator to obtain m, and returns m.

That completes the description of how an ideal channel is implemented using a channel simulator.

Now we define a notion of security based on the indistinguishability of real and ideal channels for a public-key encryption scheme PKE with an unrestricted message space. Consider a game in which a polynomial-time probabilistic adversary A interacts with an arbitrary number of channels and at the end of the game outputs a 0 or 1. We say that PKE is *secure in the sense of channel indistinguishability* if there exists an efficient *channel simulator* such that, for the resulting ideal channel, A cannot effectively distinguish between a game played with all real channels and a game played with all ideal channels; i.e., the absolute difference between the probabilities that A outputs a 1 in the two games grows negligibly in the security parameter.

Note that since real channels never become broken, this definition of security implies that ideal channels become broken with only negligible probability.

It is straightforward to show that if PKE is secure against adaptive chosen ciphertext attack, then it is also secure in the sense of channel indistinguishability. To prove this, the channel simulator is implemented using the KeyGen and Decrypt algorithms of PKE, and the Encrypt operation of the channel simulator on input ℓ simply runs the Encrypt algorithm of PKE on input 1^ℓ. It can be shown using a standard "hybrid" argument that the resulting ideal channel is indistinguishable from the real channel.

In analyzing a higher-level protocol, one may substitute all real channels by ideal channels. Presumably, it is much more straightforward to then analyze the resulting idealized protocol, since in the idealized protocol ciphertexts are just "receipts" that are *completely independent* of the corresponding messages. Security implies that any (polynomial-time recognizable) event in the original protocol occurs with essentially the same probability as in the idealized protocol.

3.4. Further discussion. The definition of security we have presented here is from [50]. It is called *IND-CCA*2 in [8]. It is known to be equivalent to other notions, such as nonmalleability [27, 8, 28], which is called *NM-CCA*2 in [8].

There are other, weaker notions of security for a public-key encryption scheme. For example, [47] defines a notion that is sometimes called *security against indifferent chosen ciphertext attack*, or *security against lunchtime attack*. This definition of security is exactly the same as the one above in section 3.2, except that Stage 4 of the attack is omitted—that is, the adversary does not have access to the decryption

oracle after it obtains the target ciphertext. While this notion of security may seem natural, it is actually not sufficient in many applications; in particular, in the channel model discussed above in section 3.3 one could not allow the interleaving of Encrypt and Decrypt operations. This notion is called *IND-CCA*1 in [8].

An even weaker notion of security for a public-key encryption scheme is that of *security against a passive attack*, also known as *semantic security*. This definition of security is exactly the same as the one above in section 3.2, except that both Stages 2 and 4 of the attack are omitted— that is, the adversary does not have access to the decryption oracle at all. This notion was introduced in [34] and is called IND-CPA in [8]. This notion of security is quite limited: it is only adequate in situations where the adversary has only the power to eavesdrop network traffic but cannot modify network traffic or otherwise actively participate in a protocol using the encryption scheme.

For a similar, but slightly different, approach to modeling encryption as an "idealized" process, see [18]. See also [7] for another generalization of the definition of adaptive chosen ciphertext attack to a setting involving many users and messages.

Another notion of security is that of *plaintext awareness*, introduced in [10] and further refined in [8], where in a certain sense an adversary who submits a ciphertext for decryption must already "know" the corresponding plaintext, and hence the decryption oracle does not really help. The notion defined in these papers makes sense only in the random oracle model, but in that model security in the sense of [8] implies security against adaptive chosen ciphertext attack. Conversely, while security against adaptive chosen ciphertext attack does not imply that an adversary "knows" the plaintext corresponding to a submitted ciphertext, it does imply that it does not "hurt" to tell him the plaintext, and in this sense the notion of security against adaptive chosen ciphertext attack is probably just as good as any notion of plaintext awareness.

4. Intractability assumptions related to the discrete logarithm problem. In this section, we recall the discrete logarithm (DL) assumption, the computational Diffie–Hellman (CDH) assumption, and the decisional Diffie–Hellman (DDH) assumption. All of these assumptions are formulated with respect to a suitable group G of large prime order q generated by a given element g.

Informally, the DL assumption is this:

given g^x, it is hard to compute x.

Informally, the CDH assumption is this:

given g^x and g^y for random $x, y \in \mathbf{Z}_q$, it is hard to compute g^{xy}.

Informally, the DDH assumption is this:

it is hard to distinguish triples of the form (g^x, g^y, g^z) for random $x, y, z \in \mathbf{Z}_q$ from triples of the form (g^x, g^y, g^{xy}) for random $x, y \in \mathbf{Z}_q$.

It is clear that the DDH assumption is at least as strong as the CDH assumption, which in turn is at least as strong as the DL assumption. The rest of this section is devoted to describing these assumptions more formally, discussing appropriate groups, and discussing some variations and consequences of these assumptions.

4.1. Computational group schemes. To state these intractability assumptions in a general but precise way and in an appropriate asymptotic setting, we introduce the notion of a *computational group scheme*.

A computational group scheme $\mathcal{G}$ specifies a *sequence* $(\mathbf{S}_\lambda)_{\lambda \in \mathbf{Z}_{\geq 0}}$ *of group distributions*. For every value of a security parameter $\lambda \in \mathbf{Z}_{\geq 0}$, $\mathbf{S}_\lambda$ is a probability distribution of group descriptions. A group description Γ specifies a finite abelian

group $\hat{G}$, along with a prime order subgroup G, a generator g of G, and the order q of G. We use multiplicative notation for the group operation in $\hat{G}$, and we denote the identity element of $\hat{G}$ by 1_G.

We will write $\Gamma[\hat{G}, G, g, q]$ to indicate that Γ specifies $\hat{G}$, G, g, and q as above. The following is a simple example of this notation: "for all $\lambda \in \mathbf{Z}_{\geq 0}$, for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we have $g^q = 1_G$."

As usual, mathematical objects such as a group description Γ and elements of a group $\hat{G}$ are represented for computational purposes as bit strings bounded in length by a polynomial in λ. The interpretation of these bit strings is up to the algorithms comprising the group scheme (see below). However, we require that the encoding scheme used to represent group elements as bit strings be *canonical*; that is, every element of a group $\hat{G}$ has a *unique* binary encoding.

The group scheme should also provide several algorithms:

- a deterministic, polynomial-time algorithm for computing the group operation that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, along with $h_1, h_2 \in \hat{G}$, and outputs the group element $h_1 \cdot h_2 \in \hat{G}$;
- a deterministic, polynomial-time algorithm for computing the group inversion operation that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, and $h \in \hat{G}$ and outputs $h^{-1} \in \hat{G}$;
- a deterministic, polynomial-time algorithm that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, and $\alpha \in \{0,1\}^*$ and determines if α is a valid binary encoding of an element of $\hat{G}$;
- a deterministic, polynomial-time algorithm that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, and $h \in \hat{G}$ and determines if $h \in G$;
- a deterministic, polynomial-time algorithm that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$ and $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$ and outputs g and q;
- a probabilistic, polynomial-time approximate sampling algorithm $\hat{S}$ that on input 1^λ approximately samples $\mathbf{S}_\lambda$: the distributions $\mathbf{S}_\lambda$ and $\hat{S}(1^\lambda)$ should be statistically close; that is, the statistical distance $\Delta(\mathbf{S}_\lambda, \hat{S}(1^\lambda))$ should be a negligible function in λ.

Notice that we do not require that the output distribution $\hat{S}(1^\lambda)$ of the sampling algorithm is identical to $\mathbf{S}_\lambda$ but only that the distributions have a negligible statistical distance. In particular, not all elements of $[\hat{S}(1^\lambda)]$ are necessarily valid group descriptions. It would be impractical to require that these two distributions be identical.

Note that the requirement that the group order be easily computable from the group description is not a trivial requirement: it is easy to exhibit groups whose orders are not easy to compute, e.g., subgroups of $\mathbf{Z}_n^*$ for composite n.

The requirement that group elements have unique encodings is also an important, nontrivial requirement. It is easy to exhibit quotient groups in which the problem of computing canonical representatives of residue classes is nontrivial. An example of this is the group underlying Paillier's encryption scheme [49].

Let $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$. The value 1_G may be directly encoded in Γ, but if not we can always compute it as $g \cdot g^{-1}$.

Although we will not require it, typical group schemes will have the property that for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$ the only elements of $\hat{G}$ of order q lie in G. When this is the case, testing whether a given $h \in \hat{G}$ lies in the subgroup G can be implemented by testing if $h^q = 1_G$. However, a group scheme may provide a more efficient subgroup test.

Let $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$. For $a \in G \setminus \{1_G\}$ and $b \in G$, we denote by $\log_a b$ the discrete logarithm of b to the base a; that is, $\log_a b$ is the unique element $x \in \mathbf{Z}_q$ such that $b = a^x$.

As a notational convention, throughout this paper, the letters a–h (and decorated versions thereof) will denote elements of $\hat{G}$, and the letters r–z (and decorated versions thereof) will denote elements of $\mathbf{Z}_q$.

4.2. Examples of appropriate computational group schemes. There are several examples of computational group schemes that are appropriate for cryptographic applications.

Example 1. Let $\ell_1(\lambda)$ and $\ell_2(\lambda)$ be polynomially bounded integer-valued functions in λ such that $1 < \ell_1(\lambda) < \ell_2(\lambda)$ for all $\lambda \in \mathbf{Z}_{\geq 0}$. It should be the case that the function $2^{-\ell_1(\lambda)}$ is negligible. For a given $\lambda \in \mathbf{Z}_{\geq 0}$, the distribution $\mathbf{S}_\lambda$ is defined as the distribution of triples (q, p, g), where

- q is a random $\ell_1(\lambda)$-bit prime,
- p is a random $\ell_2(\lambda)$-bit prime with $p \equiv 1 \pmod{q}$, and
- g is a random generator of G, the unique subgroup of order q of the cyclic group $\hat{G} = \mathbf{Z}_p^*$.

Elements in $\mathbf{Z}_p^*$ can be encoded canonically as bit strings of length $\ell_2(\lambda)$. Group operations in $\mathbf{Z}_p^*$ are efficiently implemented using arithmetic modulo p, and group inversion is implemented using the extended Euclidean algorithm. To test if an element $(\alpha \bmod p) \in \mathbf{Z}_p^*$ lies in G, we can test if $\alpha^q \equiv 1 \pmod{p}$.

A random generator g of G may be obtained by generating a random element in $\mathbf{Z}_p^*$ and raising it to the power $(p-1)/q$ (repeating if necessary if this yields $(1 \bmod p)$).

The sampling algorithm $\hat{S}$ may use standard, practical algorithms for primality testing that may err with a small probability that grows negligibly in λ. See, e.g., [4] for more information on primality testing. Not all elements of $[\hat{S}(1^\lambda)]$ are valid group descriptions. Moreover, depending on other aspects of the implementation, the distribution on the valid group descriptions may also be slightly skewed away from $\mathbf{S}_\lambda$. In our formulation of various intractability assumptions, it is much more convenient to work with the natural distribution $\mathbf{S}_\lambda$ than with the more awkward distribution $\hat{S}(1^\lambda)$.

We should comment that the density of primes p such that $p \equiv 1 \pmod{q}$ has never been proven to be sufficiently large to ensure fast termination of the group generation algorithm. Dirichlet's theorem on primes in arithmetic progressions applies only to the case where q is fixed relative to p. However, provided $\ell_2(\lambda) \geq (2 + \delta)\ell_1(\lambda)$ for some fixed $\delta > 0$, for any $\ell_1(\lambda)$-bit prime q, the probability that a random $\ell_2(\lambda)$-bit number of the form $qk + 1$ is prime is $\Omega(1/\ell_2(\lambda))$, assuming the extended Riemann hypothesis (ERH). This follows from Theorem 8.8.18 in [4].

If the density of primes p such that $p \equiv 1 \pmod{q}$ cannot be proven to be sufficiently large to ensure fast termination of the group generation algorithm, even assuming the ERH, it may not be unreasonable to still conjecture that this is the case.

Example 2. This is the same as Example 1, except that $p = 2q + 1$, where q is a random $\ell_1(\lambda)$-bit prime. Such a prime q is known as a Sophie Germain prime. It is unknown if there exist infinitely many Sophie Germain primes. However, it is conjectured that there are, and specific conjectures on their density have been made [5, 6] that empirically seem to be valid. In particular, it is conjectured that the probability that a random $\ell_1(\lambda)$-bit number is a Sophie Germain prime is $\Omega(1/\ell_1(\lambda)^2)$. If such a density estimate were true, then a simple trial and error method for finding Sophie Germain primes would terminate quickly. See [23] for more information on efficiently generating such primes.

Since the subgroup G of $\mathbf{Z}_p^*$ of order q is just the subgroup of quadratic residues, testing if a given element $(\alpha \bmod p) \in \mathbf{Z}_p^*$ lies in G can be performed by computing the Legendre symbol $(\alpha \mid p)$, which is generally much more efficient than computing $\alpha^q \bmod p$.

A nice property of this construction is that the numbers $\{1, \ldots, q\}$ are easily encoded as elements of G. Given $\alpha \in \{1, \ldots, q\}$, we test if $(\alpha \mid p) = 1$; if so, then we encode α as $(\alpha \bmod p) \in G$, and otherwise we encode α as $(-\alpha \bmod p)$. Given a group element $h = (\alpha \bmod p) \in G$ with $1 \leq \alpha \leq p-1$, we decode h as α if $\alpha \leq q$, and otherwise we decode h as $p - \alpha$.

This encoding scheme clearly allows us to also easily encode arbitrary bit strings of length $\ell_1(\lambda) - 1$ as elements of G.

Example 3. One can also construct G as a prime order subgroup of an elliptic curve over a finite field. Elliptic curves and their application to cryptography is a very rich field, and we refer the reader to [12] for an introduction and further references. We note here only that some of the same minor technical problems that arose above in Example 1 also arise here; namely, we note that (1) the known procedures for generating elliptic curves whose orders have a suitably large prime factor are somewhat heuristic, simply because not enough has been proven about how the order of a randomly generated elliptic curve factors into primes, and (2) it is in general not easy to encode arbitrary bit strings of a given length as points on an elliptic curve. We also note that it is fairly easy to generate elliptic curves of prime order so that we do not have to work in a subgroup; i.e., we can take $G = \hat{G}$. This is useful, as then the subgroup test becomes trivial.

4.3. Intractability assumptions.

4.3.1. The DL assumption. Let $\mathcal{G}$ be a computational group scheme specifying a sequence $(\mathbf{S}_\lambda)_{\lambda \in \mathbf{Z}_{\geq 0}}$ of group distributions.

For all probabilistic, polynomial-time algorithms A and for all $\lambda \in \mathbf{Z}_{\geq 0}$, we define the *DL advantage of* A *against* $\mathcal{G}$ *at* λ as

$$\mathsf{AdvDL}_{\mathcal{G},\mathsf{A}}(\lambda) := \Pr[\ y = x : \Gamma[\hat{G}, G, g, q] \xleftarrow{R} \mathbf{S}_\lambda;\ x \xleftarrow{R} \mathbf{Z}_q;\ y \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, g^x)\ ].$$

The DL assumption for $\mathcal{G}$ is this:

> *For every probabilistic, polynomial-time algorithm* A*, the function* $\mathsf{AdvDL}_{\mathcal{G},\mathsf{A}}(\lambda)$ *is negligible in* λ*.*

4.3.2. The CDH assumption. Let $\mathcal{G}$ be a computational group scheme specifying a sequence $(\mathbf{S}_\lambda)_{\lambda \in \mathbf{Z}_{\geq 0}}$ of group distributions.

For all probabilistic, polynomial-time algorithms A and for all $\lambda \in \mathbf{Z}_{\geq 0}$, we define the *CDH advantage of* A *against* $\mathcal{G}$ *at* λ as

$$\mathsf{AdvCDH}_{\mathcal{G},\mathsf{A}}(\lambda) := \Pr[\ c = g^{xy} : \Gamma[\hat{G}, G, g, q] \xleftarrow{R} \mathbf{S}_\lambda;\ x, y \xleftarrow{R} \mathbf{Z}_q;\ c \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, g^x, g^y)\ ].$$

The CDH assumption for $\mathcal{G}$ is this:

> *For every probabilistic, polynomial-time algorithm* A*, the function* $\mathsf{AdvCDH}_{\mathcal{G},\mathsf{A}}(\lambda)$ *is negligible in* λ*.*

For all probabilistic, polynomial-time algorithms A, for all $\lambda \in \mathbf{Z}_{\geq 0}$, and for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we define the *CDH advantage of* A *against* $\mathcal{G}$ *at* λ *given* Γ as

$$\mathsf{AdvCDH}_{\mathcal{G},\mathsf{A}}(\lambda \mid \Gamma) := \Pr[\ c = g^{xy} : x \xleftarrow{R} \mathbf{Z}_q;\ y \xleftarrow{R} \mathbf{Z}_q;\ c \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, g^x, g^y)\ ].$$

4.3.3. The DDH assumption. Let $\mathcal{G}$ be a computational group scheme specifying a sequence $(\mathbf{S}_\lambda)_{\lambda\in\mathbf{Z}_{\geq 0}}$ of group distributions.

For all $\lambda \in \mathbf{Z}_{\geq 0}$ and for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we define the sets $\mathcal{D}_{\lambda,\Gamma}$ and $\mathcal{T}_{\lambda,\Gamma}$ as follows:

$$\begin{aligned}\mathcal{D}_{\lambda,\Gamma} &:= \{(g^x, g^y, g^{xy}) \in G^3 : x, y \in \mathbf{Z}_q\};\\ \mathcal{T}_{\lambda,\Gamma} &:= G^3.\end{aligned}$$

The set $\mathcal{D}_{\lambda,\Gamma}$ is the set of "Diffie–Hellman triples." Also, for $\rho \in G^3$, define $\mathsf{DHP}_{\lambda,\Gamma}(\rho) = 1$ if $\rho \in \mathcal{D}_{\lambda,\Gamma}$, and otherwise define $\mathsf{DHP}_{\lambda,\Gamma}(\rho) = 0$.

For all 0/1-valued, probabilistic, polynomial-time algorithms A and for all $\lambda \in \mathbf{Z}_{\geq 0}$, we define the *DDH advantage of* A *against* $\mathcal{G}$ *at* λ as

$$\begin{aligned}&\mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}}(\lambda)\\ &:= \Big|\Pr[\ \tau = 1 : \Gamma \xleftarrow{R} \mathbf{S}_\lambda;\ \rho \xleftarrow{R} \mathcal{D}_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho)\ ]\\ &\quad - \Pr[\ \tau = 1 : \Gamma \xleftarrow{R} \mathbf{S}_\lambda;\ \rho \xleftarrow{R} \mathcal{T}_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho)\ ]\Big|.\end{aligned}$$

The DDH assumption for $\mathcal{G}$ is this:

> *For every probabilistic, polynomial-time,* 0/1*-valued algorithm* A*, the function* $\mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}}(\lambda)$ *is negligible in* λ*.*

For all 0/1-valued, probabilistic, polynomial-time algorithms A, for all $\lambda \in \mathbf{Z}_{\geq 0}$, and for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we define the *DDH advantage of* A *against* $\mathcal{G}$ *at* λ *given* Γ as

$$\begin{aligned}&\mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}}(\lambda \mid \Gamma)\\ &:= \Big|\Pr[\ \tau = 1 : \rho \xleftarrow{R} \mathcal{D}_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho)\ ]\\ &\quad - \Pr[\ \tau = 1 : \rho \xleftarrow{R} \mathcal{T}_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho)\ ]\Big|.\end{aligned}$$

A minor variation. We will need the following variation on the DDH assumption.

For all $\lambda \in \mathbf{Z}_{\geq 0}$ and for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we define the sets $\mathcal{D}'_{\lambda,\Gamma}$ and $\mathcal{T}'_{\lambda,\Gamma}$ as follows:

$$\begin{aligned}\mathcal{D}'_{\lambda,\Gamma} &:= \{g^x, g^y, g^{xy} : x, y \in \mathbf{Z}_q, x \neq 0\};\\ \mathcal{T}'_{\lambda,\Gamma} &:= \{g^x, g^y, g^z : x, y, z \in \mathbf{Z}_q, x \neq 0, z \neq xy\}.\end{aligned}$$

That is, $\mathcal{D}'_{\lambda,\Gamma}$ is the set of triples $(\hat{g}, a, \hat{a}) \in G^3$ such that $\hat{g} \neq 1_G$ and $\log_g a = \log_{\hat{g}} \hat{a}$, and $\mathcal{T}'_{\lambda,\Gamma}$ is the set of triples $(\hat{g}, a, \hat{a}) \in G^3$ such that $\hat{g} \neq 1_G$ and $\log_g a \neq \log_{\hat{g}} \hat{a}$.

It is easy to verify the following:

$$\Delta(\mathbf{U}(\mathcal{D}_{\lambda,\Gamma}), \mathbf{U}(\mathcal{D}'_{\lambda,\Gamma})) \leq 1/q; \tag{4.1}$$

$$\Delta(\mathbf{U}(\mathcal{T}_{\lambda,\Gamma}), \mathbf{U}(\mathcal{T}'_{\lambda,\Gamma})) \leq 2/q. \tag{4.2}$$

For all 0/1-valued, probabilistic, polynomial-time algorithms A and for all $\lambda \in \mathbf{Z}_{\geq 0}$, we define

$$\begin{aligned}&\mathsf{AdvDDH}'_{\mathcal{G},\mathsf{A}}(\lambda)\\ &:= \Big|\Pr[\ \tau = 1 : \Gamma \xleftarrow{R} \mathbf{S}_\lambda;\ \rho \xleftarrow{R} \mathcal{D}'_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho)\ ]\\ &\quad - \Pr[\ \tau = 1 : \Gamma \xleftarrow{R} \mathbf{S}_\lambda;\ \rho \xleftarrow{R} \mathcal{T}'_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho)\ ]\Big|.\end{aligned}$$

For all 0/1-valued, probabilistic, polynomial-time algorithms A, for all $\lambda \in \mathbf{Z}_{\geq 0}$, and for all $\Gamma \in [\mathbf{S}_\lambda]$, we define

$$\begin{aligned}&\mathsf{AdvDDH}'_{\mathcal{G},\mathsf{A}}(\lambda \mid \Gamma)\\ &:= \Big|\Pr[\ \tau = 1 : \rho \xleftarrow{R} \mathcal{D}'_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho)\ ]\\ &\qquad - \Pr[\ \tau = 1 : \rho \xleftarrow{R} \mathcal{T}'_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho)\ ]\Big|.\end{aligned}$$

The inequalities (4.1) and (4.2) imply the following.

LEMMA 4.1. *For all 0/1-valued, probabilistic, polynomial-time algorithms* A*, for all* $\lambda \in \mathbf{Z}_{\geq 0}$*, and for all* $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$*,*

$$\big|\mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}}(\lambda \mid \Gamma) - \mathsf{AdvDDH}'_{\mathcal{G},\mathsf{A}}(\lambda \mid \Gamma)\big| \leq 3/q.$$

In particular, the DDH assumption holds for $\mathcal{G}$ *if and only if for every probabilistic, polynomial-time* 0/1*-valued algorithm* A *the function* $\mathsf{AdvDDH}'_{\mathcal{G},\mathsf{A}}(\lambda)$ *is negligible in* λ*.*

Random self-reducibility. In this section, we discuss the random self-reducibility property of the DDH problem and its implications.

The following lemma states the random self-reducibility property for the DDH problem.

LEMMA 4.2. *There exists a probabilistic, polynomial-time algorithm* RSR *such that for all* $\lambda \in \mathbf{Z}_{\geq 0}$*, for all* $\Gamma \in [\mathbf{S}_\lambda]$*, and for all* $\rho \in \mathcal{T}_{\lambda,\Gamma}$*, the distribution* $\mathsf{RSR}(1^\lambda, \Gamma, \rho)$ *is* $\mathbf{U}(\mathcal{D}_{\lambda,\Gamma})$ *if* $\rho \in \mathcal{D}_{\lambda,\Gamma}$ *and is* $\mathbf{U}(\mathcal{T}_{\lambda,\Gamma})$ *if* $\rho \notin \mathcal{D}_{\lambda,\Gamma}$*.*

This was first observed by Stadler [61], who needed the result to prove the security of a particular protocol, and later by Naor and Reingold [45], who also pointed out some of its broader implications.

The algorithm RSR is very simple. Given 1^λ, the group description $\Gamma[\hat{G}, G, g, q]$, and $\rho = (a, b, c) \in G^3$, the algorithm computes $(a', b', c') \in G^3$ as follows:

$$r \xleftarrow{R} \mathbf{Z}_q;\ s \xleftarrow{R} \mathbf{Z}_q;\ t \xleftarrow{R} \mathbf{Z}_q;\ a' \leftarrow a^r g^s;\ b' \leftarrow b g^t;\ c' \leftarrow c^r a^{rt} b^s g^{st}.$$

The implication of this random self-reduction is that if Diffie–Hellman tuples can be efficiently distinguished from random tuples with a nonnegligible advantage, then Diffie–Hellman tuples can be efficiently *recognized* with negligible error probability. More formally, we have the following.

LEMMA 4.3. *For every* 0/1*-valued, probabilistic, polynomial-time algorithm* A *and every polynomial* P *(with integer coefficients, taking positive values on* $\mathbf{Z}_{\geq 0}$*), there exists a* 0/1*-valued, probabilistic, polynomial-time algorithm* A_1 *such that for all* $\lambda \in \mathbf{Z}_{\geq 0}$*, for all* $\Gamma \in [\mathbf{S}_\lambda]$*, for all* $\rho \in \mathcal{T}_{\lambda,\Gamma}$*, and for all* $\kappa \in \mathbf{Z}_{\geq 0}$*,*

if $\mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}}(\lambda \mid \Gamma) \geq 1/P(\lambda)$*, then*

$$\Pr[\tau \neq \mathsf{DHP}_{\lambda,\Gamma}(\rho) : \tau \xleftarrow{R} \mathsf{A}_1(1^\lambda, \Gamma, \rho, 1^\kappa)] \leq 2^{-\kappa}.$$

Lemma 4.3 follows from Lemma 4.2 using standard "amplification" techniques, making use of standard results on tail inequalities for the binomial distribution (cf. section C.5 in [21]). Given 1^λ, Γ, ρ, and 1^κ, algorithm A_1 invokes algorithm A as a subroutine $O(P(\lambda)^2\kappa)$ times with inputs $(1^\lambda, \Gamma, \rho')$, where each $\rho' \in \mathcal{T}_{\lambda,\Gamma}$ is independently sampled from $\mathsf{RSR}(1^\lambda, \Gamma, \rho)$; additionally, algorithm A_1 has to run algorithm A as a subroutine $O(P(\lambda)^2\kappa)$ times to "calibrate" A, calculating an estimate of

$$\Pr[\ \tau = 1 : \rho' \xleftarrow{R} \mathcal{T}_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho')\ ].$$

4.4. Further discussion. The CDH assumption was introduced informally by [26]. Since then, there have been many papers that deal with the DL and CDH assumptions and cryptographic applications based on them. The DDH assumption appears to have first surfaced in the cryptographic literature in [16], although, as that paper notes, the DDH assumption is actually needed to prove the security of a number of previously proposed protocols. Indeed, the famous Diffie–Hellman key exchange cannot be proved secure in any reasonable and standard way just based on the CDH assumption: the DDH assumption (or some variant thereof) is required.

The DDH assumption underpins a number of cryptographic applications. See, for example, the work of Stadler [61] on publicly verifiable secret sharing and the construction by Naor and Reingold [45] of pseudorandom functions. Also, the well-known encryption scheme of ElGamal [30] relies on the DDH for its security against passive attacks (i.e., semantic security).

One variant of the ElGamal scheme is as follows. Let G be a group of prime order q generated by an element g. The public key consists of a group element $h = g^z$, where $z \in \mathbf{Z}_q$ is chosen at random; the secret key is z. To encrypt a message m, where we assume that $m \in G$, we compute

$$u \stackrel{R}{\leftarrow} \mathbf{Z}_q;\ a \leftarrow g^u;\ b \leftarrow h^u;\ c \leftarrow b \cdot m$$

to form a ciphertext $\psi = (a, c)$. To decrypt such a ciphertext using the secret key, one computes

$$b \leftarrow a^z;\ m \leftarrow c \cdot b^{-1}$$

to obtain the message m.

It is easy to show that the security of this encryption scheme against passive attack is equivalent to the DDH assumption. It is also easy to see that this scheme is completely insecure against adaptive chosen ciphertext attack: if (a, c) is an encryption of $m \in G$, then for any $m' \in G$, $(a, c \cdot m')$ is an encryption of $m \cdot m'$; thus, one can submit $(a, c \cdot m')$ to the decryption oracle, obtaining $m \cdot m'$, from which one then computes m.

There are some very special families of elliptic curves for which the DDH assumption does not hold but for which the CDH assumption still appears to stand [38]. How these results are to be interpreted is a bit unclear. On the one hand, perhaps they cast some doubt on the DDH assumption in general. On the other hand, perhaps they illustrate only that very specially crafted families of elliptic curves may exhibit some surprising security weaknesses, which would seem to counsel against using such special families of elliptic curves for cryptographic applications and instead to use generic, randomly generated elliptic curves; indeed, for another special class of elliptic curves, the DL assumption is false [60].

We refer the reader to two excellent surveys, [43] and [14]. The latter focuses exclusively on the DDH assumption, while the former discusses both the CDH and the DDH assumptions. Also see [53], where it is shown that the DDH is hard in a "generic" model of computation.

5. Target collision resistant hash functions. In this section, we define the notion of a *target collision resistant hash function*, which is a special kind of *universal one-way hash function*, tailored somewhat for our particular application.

We informally summarize this section as follows. We shall be working with a group G of order q, and we want to hash tuples of group elements to elements of $\mathbf{Z}_q$.

For this purpose, we will use a family of keyed hash functions such that given a randomly chosen tuple of group elements and randomly chosen hash function key, it is computationally infeasible to find a different tuple of group elements that hashes to the same value using the given hash key.

5.1. Definitions. Let k be a fixed positive integer, and let $\mathcal{G}$ be a computational group scheme specifying a sequence $(\mathbf{S}_\lambda)_{\lambda \in \mathbf{Z}_{\geq 0}}$ of group distributions.

A *k-ary group hashing scheme* HF *associated with* $\mathcal{G}$ specifies two items:

- A family of *key spaces* indexed by $\lambda \in \mathbf{Z}_{\geq 0}$ and $\Gamma \in [\mathbf{S}_\lambda]$. Each such key space is a probability space on bit strings denoted by $\mathsf{HF.KeySpace}_{\lambda,\Gamma}$.
  There must exist a probabilistic, polynomial-time algorithm whose output distribution on input 1^λ and Γ is equal to $\mathsf{HF.KeySpace}_{\lambda,\Gamma}$.
- A family of *hash functions* indexed by $\lambda \in \mathbf{Z}_{\geq 0}$, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, and $\mathsf{hk} \in [\mathsf{HF.KeySpace}_{\lambda,\Gamma}]$, where each such function $\mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}$ maps a k-tuple $\rho \in G^k$ of group elements to an element of $\mathbf{Z}_q$.
  There must exist a deterministic, polynomial-time algorithm that on input 1^λ, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, $\mathsf{hk} \in [\mathsf{HF.KeySpace}_{\lambda,\Gamma}]$, and $\rho \in G^k$ outputs $\mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\rho)$.

Let A be a probabilistic, polynomial-time algorithm. For $\lambda \in \mathbf{Z}_{\geq 0}$, we define

$$\begin{aligned}&\mathsf{AdvTCR}_{\mathsf{HF},\mathsf{A}}(\lambda)\\ &:= \Pr[\ \rho \in G^k \ \wedge\ \rho \neq \rho^* \ \wedge\ \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\rho^*) = \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\rho) :\\ &\qquad \Gamma[\hat{G}, G, g, q] \xleftarrow{R} \mathbf{S}_\lambda;\ \rho^* \xleftarrow{R} G^k;\ \mathsf{hk} \xleftarrow{R} \mathsf{HF.KeySpace}_{\lambda,\Gamma};\ \rho \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho^*, \mathsf{hk})\ ].\end{aligned}$$

The target collision resistance (TCR) assumption for HF is this:

For every probabilistic, polynomial-time algorithm A*, the function* $\mathsf{AdvTCR}_{\mathsf{HF},\mathsf{A}}(\lambda)$ *is negligible in* λ*.*

It will also be convenient to define the following. Let A be a probabilistic, polynomial-time algorithm. For $\lambda \in \mathbf{Z}_{\geq 0}$ and $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we define

$$\begin{aligned}&\mathsf{AdvTCR}_{\mathsf{HF},\mathsf{A}}(\lambda \mid \Gamma)\\ &:= \Pr[\ \rho \in G^k \ \wedge\ \rho \neq \rho^* \ \wedge\ \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\rho^*) = \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\rho) :\\ &\qquad \rho^* \xleftarrow{R} G;\ \mathsf{hk} \xleftarrow{R} \mathsf{HF.KeySpace}_{\lambda,\Gamma};\ \rho \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \rho^*, \mathsf{hk})\ ].\end{aligned}$$

5.2. Further discussion. As already mentioned, our notion of a target collision resistant hash function is a special case of the more general notion of a universal one-way hash function, introduced by Naor and Yung [46]. In their presentation, the hash functions mapped bit strings to bit strings, but of course, using appropriate formatting, we can easily make such a function a map from tuples of elements of the group G to elements of $\mathbf{Z}_q$. The notion of security presented in [46] was also slightly stronger than ours: in their paper, the first input to the hash function (i.e., the "target" input) is chosen adversarially, but independent of the key of the hash function, whereas in our application the target input is a random tuple of group elements. Note that our usage of the term "target collision resistance" differs from that in [11], where it is used to mean a "nonasymptotic" version of security, but is otherwise identical to the notion of security for universal one-way hash functions.

As was shown in [46], universal one-way hash functions can be built from arbitrary one-way permutations. This result was extended by Rompel [51], who showed that universal one-way hash functions can be built (albeit less efficiently) from arbitrary one-way functions.

In practice, to build a universal one-way hash function one can use a dedicated cryptographic hash function, such as SHA-1 [52]. Constructions in [11] and [55] show how to build a general-purpose universal one-way hash function using the underlying compression function of SHA-1, assuming the latter is second preimage collision resistant. In practice, one might simply use SHA-1 directly without a key—it is not unreasonable to assume that this already satisfies our definition of target collision resistance.

Note that the notion of target collision resistance is both qualitatively and quantitatively weaker than the notion of (full) collision resistance, which is why we prefer to rely on the former rather than the latter. A *collision resistant* hash function is one where it is hard for an adversary to find two different inputs that hash to the same value; the difference between our notion of target collision resistance and collision resistance is that in the former, one of the two inputs is not under the control of the adversary, while in the latter both inputs are under the control of the adversary. Simon [59], in fact, gives a kind of separation result, which suggests that collision resistance is a strictly stronger notion of security than target collision resistance.

6. The new encryption scheme: Basic version.

6.1. Description of the scheme. In this section, we present the basic version, CS1, of our new scheme.

The scheme makes use of a computational group scheme $\mathcal{G}$ as described in section 4.1, defining a sequence $(\mathbf{S}_\lambda)_{\lambda \in \mathbf{Z}_{\geq 0}}$ of distributions of group descriptions and providing a sampling algorithm $\hat{S}$, where the output distribution $\hat{S}(1^\lambda)$ closely approximates $\mathbf{S}_\lambda$.

The scheme also makes use of a group hashing scheme HF associated with $\mathcal{G}$, as described in section 5.

The scheme is described in detail in Figure 6.1.

Remark 6.1. Note that this encryption scheme has a *restricted message space*: messages are elements of the group G. This limits to some degree the applicability of the scheme and the choice of group scheme; indeed, if one wants to encrypt arbitrary bit strings of some bounded length, then among the examples of group schemes discussed in section 4.2 only Example 2, based on Sophie Germain primes, is suitable.

Remark 6.2. Note that in step **D2** of the decryption algorithm, we test if a, $\hat{a}$, and c belong to the subgroup G. This test is essential to the security of the scheme. Although some group schemes may provide a more efficient method for performing these tests, in a typical implementation one may have to compute a^q, $\hat{a}^q$, and c^q, testing that each of these is 1_G.

Remark 6.3. Note that the key generation algorithm samples a group description Γ from $\hat{S}(1^\lambda)$. However, in describing the encryption scheme we assume that Γ is a valid group description. With negligible probability (in λ), Γ may not be a valid group description, in which case the behavior of the key generation, encryption, and decryption algorithms is implementation dependent.

Remark 6.4. It is straightforward to verify that this encryption scheme satisfies the basic requirements that any public-key encryption scheme should satisfy, as described in section 3.1. In particular, the *soundness* property will always hold when Γ is a valid group description.

Remark 6.5. Technically speaking, the output ψ of the encryption algorithm is actually a canonical binary encoding of the 4-tuple $(a, \hat{a}, c, d) \in G^4$. In particular, it is critical that for any two ciphertexts $\psi' \neq \psi$ the parsing algorithm in step **D1** of

Key Generation: On input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, compute

$$\Gamma[\hat{G}, G, g, q] \xleftarrow{R} \hat{S}(1^\lambda);\ \mathsf{hk} \xleftarrow{R} \mathsf{HF.KeySpace}_{\lambda,\Gamma};$$
$$w \xleftarrow{R} \mathbf{Z}_q^*;\ x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{R} \mathbf{Z}_q;$$
$$\hat{g} \leftarrow g^w;\ e \leftarrow g^{x_1}\hat{g}^{x_2};\ f \leftarrow g^{y_1}\hat{g}^{y_2};\ h \leftarrow g^{z_1}\hat{g}^{z_2}$$

and output the public key $\mathsf{PK} = (\Gamma, \mathsf{hk}, \hat{g}, e, f, h)$ and the secret key $\mathsf{SK} = (\Gamma, \mathsf{hk}, x_1, x_2, y_1, y_2, z_1, z_2)$.

Encryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a public key

$$\mathsf{PK} = (\Gamma[\hat{G}, G, g, q], \mathsf{hk}, \hat{g}, e, f, h) \in [\mathbf{S}_\lambda] \times [\mathsf{HF.KeySpace}_{\lambda,\Gamma}] \times G^4,$$

along with a message $m \in G$, compute

E1: $u \xleftarrow{R} \mathbf{Z}_q$;
E2: $a \leftarrow g^u$;
E3: $\hat{a} \leftarrow \hat{g}^u$;
E4: $b \leftarrow h^u$;
E5: $c \leftarrow b \cdot m$;
E6: $v \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a}, c)$;
E7: $d \leftarrow e^u f^{uv}$

and output the ciphertext $\psi = (a, \hat{a}, c, d)$.

Decryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a secret key

$$\mathsf{SK} = (\Gamma[\hat{G}, G, g, q], \mathsf{hk}, x_1, x_2, y_1, y_2, z_1, z_2) \in [\mathbf{S}_\lambda] \times [\mathsf{HF.KeySpace}_{\lambda,\Gamma}] \times \mathbf{Z}_q^6,$$

along with a ciphertext ψ, do the following.

D1: Parse ψ as a 4-tuple $(a, \hat{a}, c, d) \in \hat{G}^4$; output reject and halt if ψ is not of this form.
D2: Test if a, $\hat{a}$, and c belong to G; output reject and halt if this is not the case.
D3: Compute $v \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a}, c)$.
D4: Test if $d = a^{x_1+y_1 v} \cdot \hat{a}^{x_2+y_2 v}$; output reject and halt if this is not the case.
D5: Compute $b \leftarrow a^{z_1}\hat{a}^{z_2}$.
D6: Compute $m \leftarrow c \cdot b^{-1}$ and output m.

FIG. 6.1. *The public-key encryption scheme* CS1.

the decryption algorithm should not output the same 4-tuple of group elements.

6.2. Security analysis of the scheme. We shall prove that CS1 is secure against adaptive chosen ciphertext attack if the DDH assumption holds for $\mathcal{G}$ and the TCR assumption holds for HF. However, we wish to state and prove a concrete security reduction. To this end, we need some auxiliary definitions.

Suppose PKE is a public-key encryption scheme that uses a group scheme in the following natural way: on input 1^λ, the key generation algorithm runs the sampling algorithm of the group scheme on input 1^λ, yielding a group description Γ. For a given probabilistic, polynomial-time oracle query machine A, $\lambda \in \mathbf{Z}_{\geq 0}$, and group description Γ, let us define $\mathsf{AdvCCA}_{\mathsf{PKE},\mathsf{A}}(\lambda \mid \Gamma)$ to be A's advantage in an adaptive chosen ciphertext attack, where the key generation algorithm uses the given value of Γ instead of running the sampling algorithm of the group scheme.

For all probabilistic, polynomial-time oracle query machines A, for all $\lambda \in \mathbf{Z}_{\geq 0}$,

let $Q_{\mathsf{A}}(\lambda)$ be an upper bound on the number of decryption oracle queries made by A on input 1^λ. We assume that $Q_{\mathsf{A}}(\lambda)$ is a strict bound in the sense that it holds regardless of the probabilistic choices of A and regardless of the responses to its oracle queries from its environment.

THEOREM 6.1. *If the DDH assumption holds for $\mathcal{G}$ and the TCR assumption holds for HF, then $\mathsf{CS1}$ is secure against adaptive chosen ciphertext attack.*

In particular, for all probabilistic, polynomial-time oracle query machines A, there exist probabilistic algorithms A_1 and A_2, whose running times are essentially the same as that of A, such that the following holds. For all $\lambda \in \mathbf{Z}_{\geq 0}$ and all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we have

$$\mathsf{AdvCCA}_{\mathsf{CS1},\mathsf{A}}(\lambda \mid \Gamma) \leq \mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}_1}(\lambda \mid \Gamma) + \mathsf{AdvTCR}_{\mathsf{HF},\mathsf{A}_2}(\lambda \mid \Gamma) + (Q_{\mathsf{A}}(\lambda)+4)/q. \tag{6.1}$$

The precise running times of algorithms A_1 and A_2 depend a good deal on details of the model of computation and on implementation details, and so we make no attempt to be more precise on this matter.

Before continuing, we state the following simple but useful lemma, which we leave to the reader to verify.

LEMMA 6.2. *Let U_1, U_2, and F be events defined on some probability space. Suppose that the event $U_1 \wedge \neg F$ occurs if and only if $U_2 \wedge \neg F$ occurs. Then $|\Pr[U_1] - \Pr[U_2]| \leq \Pr[F]$.*

To prove Theorem 6.1, let us fix a probabilistic, polynomial-time oracle query machine A, the value of the security parameter $\lambda \in \mathbf{Z}_{\geq 0}$, and the group description $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$.

The attack game is as described in section 3.2. We now describe the relevant random variables to be considered in analyzing the adversary's attack.

Suppose that the public key is $(\Gamma, \mathsf{hk}, \hat{g}, e, f, h)$ and that the secret key is $(\Gamma, \mathsf{hk}, x_1, x_2, y_1, y_2, z_1, z_2)$. Let $w := \log_g \hat{g}$, and define $x, y, z \in \mathbf{Z}_q$ as follows:

$$x := x_1 + x_2 w,\ \ y := y_1 + y_2 w,\ \ z := z_1 + z_2 w.$$

That is, $x = \log_g e$, $y = \log_g f$, and $z = \log_g h$.

As a notational convention, whenever a particular ciphertext ψ is under consideration in some context, the following values are also implicitly defined in that context:

- $a, \hat{a}, b, c, d \in G$, where $\psi = (a, \hat{a}, c, d)$ and $b := a^{z_1} \hat{a}^{z_2}$;
- $u, \hat{u}, v, r, s, t \in \mathbf{Z}_q$, where

$$u := \log_g a,\ \ \hat{u} := \log_{\hat{g}} \hat{a},\ \ v := \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a}, c),\ \ r := \log_g c,\ \ s := \log_g d,$$

and

$$t := x_1 u + y_1 uv + x_2 \hat{u} w + y_2 \hat{u} v w.$$

For the target ciphertext ψ^*, we also denote $a^*, \hat{a}^*, b^*, c^*, d^* \in G$ and $u^*, \hat{u}^*, v^*, r^*, s^*, t^* \in \mathbf{Z}_q$ the corresponding values.

The probability space defining the attack game is then determined by the following, mutually independent, random variables:

- the coin tosses Coins of A;
- the values $\mathsf{hk}, w, x_1, x_2, y_1, y_2, z_1, z_2$ generated by the key generation algorithm;
- the values $\sigma \in \{0, 1\}$ and $u^* \in \mathbf{Z}_q$ generated by the encryption oracle.

Let $\mathbf{G}_0$ be the original attack game, let $\hat{\sigma} \in \{0,1\}$ denote the output of A, and let T_0 be the event that $\sigma = \hat{\sigma}$ in $\mathbf{G}_0$ so that $\mathsf{AdvCCA}_{\mathsf{CS1},\mathsf{A}}(\lambda \mid \Gamma) = |\Pr[T_0] - 1/2|$.

Our overall strategy for the proof is as follows. We shall define a sequence $\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_\ell$ of modified attack games. Each of the games $\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_\ell$ operates on the same underlying probability space. In particular, the public key and secret key of the cryptosystem, the coin tosses Coins of A, and the hidden bit σ take on *identical* values across all games. Only some of the rules defining how the environment responds to oracle queries differ from game to game. For any $1 \leq i \leq \ell$, we let T_i be the event that $\sigma = \hat{\sigma}$ in game $\mathbf{G}_i$.

To assist the reader, here is a high-level "road map" of the games:

- In game $\mathbf{G}_1$, we modify the encryption oracle so that it uses the secret key to do the encryption rather than the public key. This change is purely conceptual, and $\Pr[T_1] = \Pr[T_0]$.
- In game $\mathbf{G}_2$, we modify the encryption oracle again so that the Diffie–Hellman triple $(\hat{g}, a^*, \hat{a}^*)$ is replaced by a random triple. Under the DDH assumption, A will hardly notice, and, in particular, $|\Pr[T_2] - \Pr[T_1]|$ will be negligible.
- In game $\mathbf{G}_3$, we modify the decryption oracle so that it rejects all ciphertexts ψ such that $u \neq \hat{u}$, i.e., such that $(\hat{g}, a, \hat{a})$ is not a Diffie–Hellman triple. We will see that $\Pr[T_2]$ and $\Pr[T_3]$ differ by an amount bounded by $\Pr[R_3]$, where R_3 is the event that a ciphertext is rejected in $\mathbf{G}_3$ that would not have been under the rules of game $\mathbf{G}_2$.
- In game $\mathbf{G}_4$, we modify the encryption oracle again so that now the ciphertext is constructed without even looking at either σ, m_0, or m_1. We will see that $1/2 = \Pr[T_4] = \Pr[T_3]$ and that $\Pr[R_4] = \Pr[R_3]$ (where R_4 is defined in the same way as R_3 but with respect to game $\mathbf{G}_4$).
- In game $\mathbf{G}_5$, we add another rejection rule to the decryption oracle, this time rejecting all ciphertexts ψ such that $(a, \hat{a}, c) \neq (a^*, \hat{a}^*, c^*)$, but $v = v^*$. If A were able to produce such a ciphertext, this would represent a collision in the hash function, and so this rejection rule will be applied with negligible probability. Under the TCR assumption, this implies that $|\Pr[R_5] - \Pr[R_4]|$ is negligible (where R_5 is defined in the same way as R_3 but with respect to game $\mathbf{G}_5$). We will also see that $\Pr[R_5]$ is negligible (unconditionally).

Tracing through the above steps, one sees that $|\Pr[T_0] - 1/2|$ is negligible.

We now present the details, but we shall defer the proofs of all lemmas to the end of the proof of the theorem.

Game $\mathbf{G}_1$. We now modify game $\mathbf{G}_0$ to obtain a new game $\mathbf{G}_1$. These two games are identical, except for a small modification to the encryption oracle. Instead of using the encryption algorithm as given to compute the target ciphertext ψ^*, we use a modified encryption algorithm in which steps **E4** and **E7** are replaced by

E4′: $b \leftarrow a^{z_1} \hat{a}^{z_2}$;

E7′: $d \leftarrow a^{x_1 + y_1 v} \cdot \hat{a}^{x_2 + y_2 v}$.

The change we have made is purely conceptual: the values of b^* and d^* are exactly the same in game $\mathbf{G}_1$ as they were in $\mathbf{G}_0$. Therefore,

$$\Pr[T_1] = \Pr[T_0]. \tag{6.2}$$

Note that the encryption oracle now makes use of some components of the secret key, which is something the original encryption oracle does not do.

Game $\mathbf{G}_2$. We now modify game $\mathbf{G}_1$ to obtain a new game $\mathbf{G}_2$. We again modify the encryption oracle, replacing step **E3** of the encryption algorithm by

E3′: $\hat{u} \xleftarrow{R} \mathbf{Z}_q \setminus \{u\}$; $\hat{a} \leftarrow \hat{g}^{\hat{u}}$.

Note that whereas in games $\mathbf{G}_0$ and $\mathbf{G}_1$ we had $u^* = \hat{u}^*$, in game $\mathbf{G}_2$ u^* and $\hat{u}^*$ are nearly independent, being subject only to $u^* \neq \hat{u}^*$. However, observe that games $\mathbf{G}_1$ and $\mathbf{G}_2$ are the same, except that in game $\mathbf{G}_1$ the triple $(\hat{g}, a^*, \hat{a}^*)$ is uniformly distributed in $\mathcal{D}'_{\lambda,\Gamma}$, and in game $\mathbf{G}_2$ the triple $(\hat{g}, a^*, \hat{a}^*)$ is uniformly distributed in $\mathcal{T}'_{\lambda,\Gamma}$. Thus, any difference in behavior between these two games immediately yields a statistical test for distinguishing Diffie–Hellman triples from random triples. More precisely, we have the following.

LEMMA 6.3. *There exists a probabilistic algorithm* A_1*, whose running time is essentially the same as that of* A*, such that*

$$|\Pr[T_2] - \Pr[T_1]| \leq \mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}_1}(\lambda \mid \Gamma) + 3/q. \tag{6.3}$$

Game $\mathbf{G_3}$. In this game, we modify the *decryption* oracle in game $\mathbf{G}_2$ to obtain a new game $\mathbf{G}_3$. Instead of using the original decryption algorithm, we modify the decryption algorithm, replacing steps **D4** and **D5** by

D4′: test if $\hat{a} = a^w$ and $d = a^{x+yv}$; output reject and halt if this is not the case;

D5′: $b \leftarrow a^z$.

Note that the decryption oracle now makes use of w but does not make use of $x_1, y_2, y_1, y_2, z_1, z_2$, except indirectly through the values x, y, z.

Now, let R_3 be the event that in game $\mathbf{G}_3$ some ciphertext ψ is submitted to the decryption oracle that is rejected in step **D4′** but that would have passed the test in step **D4**.

Note that if a ciphertext passes the test in **D4′**, it would also have passed the test in **D4**.

It is clear that games $\mathbf{G}_2$ and $\mathbf{G}_3$ proceed identically until the event R_3 occurs. In particular, the events $T_2 \wedge \neg R_3$ and $T_3 \wedge \neg R_3$ are identical. So, by Lemma 6.2, we have

$$|\Pr[T_3] - \Pr[T_2]| \leq \Pr[R_3], \tag{6.4}$$

and so it suffices to bound $\Pr[R_3]$. We introduce auxiliary games $\mathbf{G}_4$ and $\mathbf{G}_5$ below to do this.

Game $\mathbf{G_4}$. This game is identical to game $\mathbf{G}_3$, except for a small modification to the *encryption* oracle. We again modify the algorithm used by the encryption oracle, replacing step **E5** by

E5′: $r \xleftarrow{R} \mathbf{Z}_q$; $c \leftarrow g^r$.

It is clear by construction that

$$\Pr[T_4] = 1/2, \tag{6.5}$$

since in game $\mathbf{G}_4$ the variable σ is never used at all, and so the adversary's output is independent of σ.

Define the event R_4 to be the event in game $\mathbf{G}_4$ analogous to the event R_3 in game $\mathbf{G}_3$; that is, R_4 is the event that in game $\mathbf{G}_4$ some ciphertext ψ is submitted to the decryption oracle that is rejected in step **D4′** but that would have passed the test in step **D4**.

We show that this modification has no effect; more precisely, we have the following.

LEMMA 6.4. *We have*

$$\Pr[T_4] = \Pr[T_3], \tag{6.6}$$

$$\Pr[R_4] = \Pr[R_3]. \tag{6.7}$$

Game $\mathbf{G_5}$. This game is the same as game $\mathbf{G}_4$, except for the following modification.

We modify the *decryption* oracle so that it applies the following *special rejection rule*: if the adversary submits a ciphertext ψ for decryption at a point in time after the encryption oracle has been invoked such that $(a, \hat{a}, c) \neq (a^*, \hat{a}^*, c^*)$ but $v = v^*$, then the decryption oracle immediately outputs reject and halts (before executing step **D4′**).

To analyze this game, we define two events.

First, we define the event C_5 to be the event that the decryption oracle in game $\mathbf{G}_5$ rejects a ciphertext using the special rejection rule.

Second, we define the event R_5 to be the event in game $\mathbf{G}_5$ that some ciphertext ψ is submitted to the decryption oracle that is rejected in step **D4′** but that would have passed the test in step **D4**. Note that such a ciphertext is not rejected by the special rejection rule, since that rule is applied before step **D4′** is executed.

Now, it is clear that games $\mathbf{G}_4$ and $\mathbf{G}_5$ proceed identically until event C_5 occurs. In particular, the events $R_4 \wedge \neg C_5$ and $R_5 \wedge \neg C_5$ are identical. So, by Lemma 6.2, we have

$$|\Pr[R_5] - \Pr[R_4]| \leq \Pr[C_5]. \tag{6.8}$$

Now, if event C_5 occurs with nonnegligible probability, we immediately get an algorithm that contradicts the target collision resistance assumption; more precisely, we have the following.

LEMMA 6.5. *There exists a probabilistic algorithm* A_2*, whose running time is essentially the same as that of* A*, such that*

$$\Pr[C_5] \leq \mathsf{AdvTCR}_{\mathsf{HF},\mathsf{A}_2}(\lambda \mid \Gamma) + 1/q. \tag{6.9}$$

Finally, we show that event R_5 occurs with negligible probability, based on purely information-theoretic considerations.

LEMMA 6.6. *We have*

$$\Pr[R_5] \leq Q_{\mathsf{A}}(\lambda)/q. \tag{6.10}$$

The detailed proof of this lemma is presented below. However, the basic idea of the proof runs as follows. For a decryption query ψ, the only information the adversary has about (x_1, x_2, y_1, y_2) are the values of x, y, and possibly s^*, which are linear combinations of (x_1, x_2, y_1, y_2). As we will prove, the value of t, which the adversary must successfully guess in order to make the event R_5 happen, is an independent linear combination of (x_1, x_2, y_1, y_2) and is therefore unpredictable.

Inequality (6.1) now follows immediately from (6.2)–(6.10).

Proofs of lemmas. To complete the proof of Theorem 6.1, we now present the proofs of Lemmas 6.3, 6.4, 6.5, and 6.6.

Proof of Lemma 6.3. We describe the algorithm A_1 in detail. For a given value of $\lambda \in \mathbf{Z}_{\geq 0}$, it takes as input 1^λ, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, and $\rho = (\hat{g}, a^*, \hat{a}^*) \in G^3$.

Algorithm A_1 provides an environment for A, interacting with A as follows.

First, A_1 computes

$$\mathsf{hk} \xleftarrow{R} \mathsf{HF.KeySpace}_{\lambda,\Gamma};\ x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{R} \mathbf{Z}_q;$$
$$e \leftarrow g^{x_1}\hat{g}^{x_2};\ f \leftarrow g^{y_1}\hat{g}^{y_2};\ h \leftarrow g^{z_1}\hat{g}^{z_2}$$

in order to generate a public key $\mathsf{PK} = (\Gamma, \mathsf{hk}, \hat{g}, e, f, h)$ and a secret key $\mathsf{SK} = (\Gamma, \mathsf{hk}, x_1, x_2, y_1, y_2, z_1, z_2)$. It then gives PK to A.

Whenever A submits a ciphertext $\psi = (a, \hat{a}, c, d)$ to the decryption oracle, A_1 simply runs the decryption algorithm, using the secret key SK.

When A submits (m_0, m_1) to the encryption oracle, A_1 computes

$$\sigma \xleftarrow{R} \{0,1\};\ b^* \leftarrow (a^*)^{z_1}(\hat{a}^*)^{z_2};\ c^* \leftarrow b^* \cdot m_\sigma;$$
$$v^* \leftarrow \mathsf{HF}^{\lambda,\Gamma}_{\mathsf{hk}}(a^*, \hat{a}^*, c^*);\ d^* \leftarrow (a^*)^{x_1 + y_1 v^*}(\hat{a}^*)^{x_2 + y_2 v^*}$$

and responds with the "ciphertext" $\psi^* = (a^*, \hat{a}^*, c^*, d^*)$.

When A outputs $\hat{\sigma}$ and halts, A_1 outputs 1 if $\sigma = \hat{\sigma}$ and 0 if $\sigma \neq \hat{\sigma}$.

That completes the description of A_1. By construction, it is clear that for fixed λ and $\Gamma \in [\mathbf{S}_\lambda]$,

$$\Pr[T_1] = \Pr[\ \tau = 1 : \rho \xleftarrow{R} \mathcal{D}'_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}_1(1^\lambda, \Gamma, \rho)\ ];$$
$$\Pr[T_2] = \Pr[\ \tau = 1 : \rho \xleftarrow{R} \mathcal{T}'_{\lambda,\Gamma};\ \tau \xleftarrow{R} \mathsf{A}_1(1^\lambda, \Gamma, \rho)\ ].$$

Thus,

$$|\Pr[T_2] - \Pr[T_1]| = \mathsf{AdvDDH}'_{\mathcal{G},\mathsf{A}_1}(\lambda \mid \Gamma),$$

and so (6.3) now follows directly from this and Lemma 4.1. □

Before continuing, we state and prove a simple but useful lemma. First, we introduce some notation: for a field K and positive integers k, n, we denote by $K^{k\times n}$ the set of all $k \times n$ matrices over K, i.e., matrices with k rows and n columns whose entries are in K; for a matrix M, we denote by M^T its transpose.

LEMMA 6.7. *Let k, n be integers with $1 \le k \le n$, and let K be a finite field. Consider a probability space with random variables $\vec{\alpha} \in K^{n\times 1}$, $\vec{\beta} = (\beta_1, \ldots, \beta_k)^T \in K^{k\times 1}$, $\vec{\gamma} \in K^{k\times 1}$, and $M \in K^{k\times n}$ such that $\vec{\alpha}$ is uniformly distributed over $K^{n\times 1}$, $\vec{\beta} = M\vec{\alpha} + \vec{\gamma}$, and for $1 \le i \le k$ the ith rows of M and $\vec{\gamma}$ are determined by $\beta_1, \ldots, \beta_{i-1}$.*

Then conditioning on any fixed values of $\beta_1, \ldots, \beta_{k-1}$ such that the resulting matrix M has rank k, the value of β_k is uniformly distributed over K in the resulting conditional probability space.

Proof. Consider fixed values of $\beta_1, \ldots, \beta_{k-1} \in K$, which determine M and $\vec{\gamma}$, and assume that the matrix M has rank k. For any $\beta_k \in K$, consider the corresponding vector $\vec{\beta} = (\beta_1, \ldots, \beta_k)^T$; there are exactly $|K|^{n-k}$ vectors $\vec{\alpha}$ such that $\vec{\beta} = M\vec{\alpha} + \vec{\gamma}$. Therefore, each possible value $\beta_k \in K$ is equally likely. □

Proof of Lemma 6.4. Consider the quantity

$$X := (\mathsf{Coins}, \mathsf{hk}, w, x_1, x_2, y_1, y_2, \sigma, u^*, \hat{u}^*)$$

and the quantity z. Note that X and z take on the same values in games $\mathbf{G}_3$ and $\mathbf{G}_4$.

Consider also the quantity r^*. This quantity takes on different values in games $\mathbf{G}_3$ and $\mathbf{G}_4$. For clarity, let us denote these values as $[r^*]_3$ and $[r^*]_4$, respectively.

It is clear by inspection that the events R_3 and T_3 are determined as functions of X, z, and $[r^*]_3$. Also, the events R_4 and T_4 have precisely the same functional dependence on X, z, and $[r^*]_4$. So to prove the lemma it suffices to show that the distributions of $(X, z, [r^*]_3)$ and $(X, z, [r^*]_4)$ are identical. Observe that by construction, conditioning on any fixed values of X and z, the distribution of $[r^*]_4$ is uniform over $\mathbf{Z}_q$. So it will suffice to show that conditioning on any fixed values of X and z, the distribution of $[r^*]_3$ is also uniform over $\mathbf{Z}_q$.

We have

$$\begin{pmatrix} z \\ [r^*]_3 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & w \\ u^* & w\hat{u}^* \end{pmatrix}}_{=:M} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \log_g m_\sigma \end{pmatrix}.$$

Conditioning only on a fixed value of X, the matrix M is fixed, but the values z_1 and z_2 are still uniformly and independently distributed over $\mathbf{Z}_q$. Observe that $\det(M) = w(\hat{u}^* - u^*) \neq 0$. If we further condition on a fixed value of z, the value of m_σ is fixed, and by Lemma 6.7 the distribution of $[r^*]_3$ is uniform over $\mathbf{Z}_q$. □

Proof of Lemma 6.5. Algorithm A_2 provides an environment for A, interacting with A as follows.

Algorithm A_2 takes as input 1^λ, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, $\rho^* = (a^*, \hat{a}^*, c^*) \in G^3$, and $\mathsf{hk} \in [\mathsf{HF.KeySpace}_{\lambda,\Gamma}]$. It first constructs a public key PK and secret key SK for the encryption scheme using the standard key generation algorithm, except that the given values of Γ and hk are used. It also constructs the target ciphertext $\psi^* = (a^*, \hat{a}^*, c^*, d^*)$, where $a^*, \hat{a}^*, c^*$ are the given inputs as above and where d^* is computed as

$$v^* \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a^*, \hat{a}^*, c^*);\ d^* \leftarrow (a^*)^{x_1 + y_1 v^*} (\hat{a}^*)^{x_2 + y_2 v^*}.$$

Here, hk is the given input as above, and x_1, y_1, x_2, y_2 are the values taken from the secret key SK as computed above.

Now A_2 interacts with A using the rules of game $\mathbf{G}_5$ for the decryption oracle and giving A the target ciphertext ψ^* when A invokes the encryption oracle. However, if the decryption oracle ever invokes the special rejection rule in game $\mathbf{G}_5$ for a given ciphertext ψ, algorithm A_2 immediately outputs $(a, \hat{a}, c)$ corresponding to ψ and halts. Also, if the attack terminates without the special rejection rule ever having been invoked, then A_2 also halts (without producing any output).

That completes the description of A_2. If the input $(a^*, \hat{a}^*, c^*)$ to A_2 is sampled uniformly over all triples of group elements subject to $\log_g a^* \neq \log_{\hat{g}} \hat{a}^*$, then algorithm A_2 succeeds in finding a collision with probability exactly $\Pr[C_5]$. However, in the definition of AdvTCR the input is sampled from the uniform distribution over all triples, not subject to the above restriction. The bound (6.9) follows from the fact that the statistical distance between these two input distributions is $1/q$. □

Proof of Lemma 6.6. To prove (6.10), for $1 \leq i \leq Q_{\mathsf{A}}(\lambda)$, let us define $R_5^{(i)}$ to be the event that there is an ith ciphertext submitted to the decryption oracle in game $\mathbf{G}_5$ and that the submitted ciphertext is rejected in step **D4′** but would have passed the test in step **D4**. For $1 \leq i \leq Q_{\mathsf{A}}(\lambda)$, let us define $B_5^{(i)}$ to be the event that the ith decryption oracle query occurs *before* the encryption oracle query and that the submitted ciphertext passes the test in steps **D1** and **D2** of the decryption oracle. For $1 \leq i \leq Q_{\mathsf{A}}(\lambda)$, let us define $\hat{B}_5^{(i)}$ to be the event that the ith decryption oracle query occurs *after* the encryption oracle query and that the submitted ciphertext passes the tests in steps **D1** and **D2** of the decryption oracle.

The bound (6.10) will follow immediately from Lemmas 6.8 and 6.9 below. □

LEMMA 6.8. *Notation is the same as in the proof of Lemma* 6.6. *For all* $1 \leq i \leq Q_{\mathsf{A}}(\lambda)$, *we have* $\Pr[R_5^{(i)} | B_5^{(i)}] \leq 1/q$.

Proof. Fix $1 \leq i \leq Q_{\mathsf{A}}(\lambda)$. Consider the quantities

$$X := (\mathsf{Coins}, \mathsf{hk}, w, z)$$

and

$$X' := (x, y).$$

The values of X and X' completely determine the behavior of the adversary up until the point when the encryption oracle is invoked, and, in particular, they completely determine the event $B_5^{(i)}$. Let us call X and X' *relevant* if the event $B_5^{(i)}$ occurs.

It will suffice to prove that conditioned on any fixed, relevant values of X and X', the probability that $R_5^{(i)}$ occurs is bounded by $1/q$.

Once relevant values of X and X' are fixed, the value ψ of the ith decryption query is also fixed, along with the corresponding values a, $\hat{a}$, b, c, d, u, $\hat{u}$, v, r, and s.

The test in **D4′** fails if and only if one of the two mutually exclusive conditions ($\hat{a} \neq a^w$) or ($\hat{a} = a^w$ and $d \neq a^{x+yv}$) holds. It is easy to verify that if the second condition holds, then in fact the test in **D4** fails. Thus, if the test in **D4′** fails but the one in **D4** passes, it must be the case that $\hat{a} \neq a^w$ and $d = a^{x_1+y_1v}\hat{a}^{x_2+y_2v}$. So we need only to consider values of X and X' such that $\hat{a} \neq a^w$. The condition $\hat{a} \neq a^w$ is equivalent to the condition $u \neq \hat{u}$, and the condition $d = a^{x_1+y_1v}\hat{a}^{x_2+y_2v}$ is equivalent to the condition $s = t$.

We have

$$\begin{pmatrix} x \\ y \\ t \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ u & \hat{u}w & uv & \hat{u}vw \end{pmatrix}}_{=: M} \cdot \begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{pmatrix}.$$

Let us first condition only on a fixed value of X, which fixes the first two rows of M but leaves the values x_1, x_2, y_1, and y_2 still uniformly distributed over $\mathbf{Z}_q$ and mutually independent. Let us further condition on a fixed value of X' such that X and X' are relevant and that $u \neq \hat{u}$. The third row of M is also fixed, along with the values x, y, and s. It is easy to see by inspection that the rows of M are linearly independent, since $\hat{u} \neq u$ and $w \neq 0$. From this, it follows by Lemma 6.7 that t is still uniformly distributed over $\mathbf{Z}_q$, but, since s is fixed, we have $\Pr[s = t] = 1/q$. □

LEMMA 6.9. *Notation is the same as in the proof of Lemma* 6.6. *For all* $1 \leq i \leq Q_{\mathsf{A}}(\lambda)$, *we have* $\Pr[R_5^{(i)} | \hat{B}_5^{(i)}] \leq 1/q$.

Proof. Fix $1 \leq i \leq Q_{\mathsf{A}}(\lambda)$. Consider the quantities

$$X := (\mathsf{Coins}, \mathsf{hk}, w, z, u^*, \hat{u}^*, r^*)$$

and

$$X' := (x, y, s^*).$$

The values of X and X' completely determine the adversary's entire behavior in game $\mathbf{G}_5$, and, in particular, they completely determine the event $\hat{B}_5^{(i)}$. Let us call X and X' *relevant* if the event $\hat{B}_5^{(i)}$ occurs.

It will suffice to prove that conditioned on any fixed, relevant values of X and X', the probability that $R_5^{(i)}$ occurs is bounded by $1/q$.

Once X and X' are fixed, the value ψ of the ith decryption query is also fixed, along with the corresponding values a, $\hat{a}$, b, c, d, u, $\hat{u}$, v, r, and s. As in the proof of Lemma 6.8, it suffices to consider values of X and X' for which $u \neq \hat{u}$ and then to

show that $\Pr[s = t] \leq q$. Notice that the value of X determines the value of v^*, and we may also assume that $v \neq v^*$. To see why we may do so, if $v = v^*$, then either $(a, \hat{a}, c) = (a^*, \hat{a}^*, c^*)$ or ψ is rejected by the special rejection rule. In the first case, since $\psi \neq \psi^*$, we must have $d \neq d^*$, but this implies that ψ fails the test in **D4**. In the second case, step **D4′** is not even executed.

We have

$$\begin{pmatrix} x \\ y \\ s^* \\ t \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ u^* & \hat{u}^* w & u^* v^* & \hat{u}^* v^* w \\ u & \hat{u} w & uv & \hat{u} v w \end{pmatrix}}_{=: M} \cdot \begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{pmatrix}.$$

Let us first condition only on a fixed value of X, which fixes the first three rows of M but leaves the values x_1, x_2, y_1, and y_2 still uniformly distributed over $\mathbf{Z}_q$ and mutually independent. Let us further condition on a fixed value of X' such that X and X' are relevant and that $u \neq \hat{u}$ and $v \neq v^*$. The fourth row of M is also fixed, along with the values x, y, s^*, and s. It is easy to see that the rows of M are linearly independent, since

$$\det(M) = w^2(\hat{u} - u)(\hat{u}^* - u^*)(v^* - v) \neq 0.$$

From this, it follows by Lemma 6.7 that t is still uniformly distributed over $\mathbf{Z}_q$, but, since s is fixed, we have $\Pr[s = t] = 1/q$. □

6.3. Two variations. Scheme CS1 was presented because it is in a form that is particularly easy to analyze. We now describe and analyze two variations of the scheme CS1, which we call CS1a and CS1b, that are a bit simpler than CS1 but require a bit more work to analyze. For both of these schemes, the public key has the same format and indeed the same probability distribution, as in CS1, and the encryption algorithm is the same as in CS1. The key generation and decryption algorithms are slightly different, however, and are described in detail in Figures 6.2 and 6.3.

Remark 6.6. Scheme CS1a is essentially the same scheme that was originally presented as the "main scheme" in [22]. Scheme CS1b is a minor variation of a scheme originally presented in [56].

Remark 6.7. Note that in scheme CS1b we do not have to separately test if $\hat{a}$ belongs to the subgroup G in step **D2′**, since this is already implied by the test in step **D4′**. The test that a and c belong to G may in some cases be implemented by testing if $a^q = 1_G$ and $c^q = 1_G$.

Remark 6.8. Note also in scheme CS1b that the decryption algorithm has to compute either three or four (if we test if $a^q = 1_G$) powers of a and possibly one power of c (if we test if $c^q = 1_G$). Special algorithmic techniques [17, 40] can be employed to compute these several powers of a significantly faster than computing several powers of different group elements.

Remark 6.9. In an actual implementation, it is strongly recommended to compute both exponentiations in step **D4′** of CS1b before rejecting the ciphertext, even if the first exponentiation performed already implies that the ciphertext should be rejected. The reason is that if the ciphertext is rejected after just one exponentiation, this may reveal some timing information that could be exploited by an attacker. Indeed, if we reject immediately upon detecting that $\hat{a} \neq a^w$, then based upon timing information an attacker could use the decryption box as a kind of Diffie–Hellman decision oracle.

Key Generation: On input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, compute

$$\Gamma[\hat{G}, G, g, q] \xleftarrow{R} \hat{S}(1^\lambda);\ \mathsf{hk} \xleftarrow{R} \mathsf{HF.KeySpace}_{\lambda,\Gamma};$$
$$w \xleftarrow{R} \mathbf{Z}_q^*;\ x_1, x_2, y_1, y_2, z \xleftarrow{R} \mathbf{Z}_q;$$
$$\hat{g} \leftarrow g^w;\ e \leftarrow g^{x_1}\hat{g}^{x_2};\ f \leftarrow g^{y_1}\hat{g}^{y_2};\ h \leftarrow g^z$$

and output the public key $\mathsf{PK} = (\Gamma, \mathsf{hk}, \hat{g}, e, f, h)$ and the secret key $\mathsf{SK} = (\Gamma, \mathsf{hk}, x_1, x_2, y_1, y_2, z)$.

Decryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a secret key

$$\mathsf{SK} = (\Gamma[\hat{G}, G, g, q], \mathsf{hk}, x_1, x_2, y_1, y_2, z) \in [\mathbf{S}_\lambda] \times [\mathsf{HF.KeySpace}_{\lambda,\Gamma}] \times \mathbf{Z}_q^5,$$

along with a ciphertext ψ, do the following.

D1: Parse ψ as a 4-tuple $(a, \hat{a}, c, d) \in \hat{G}^4$; output reject and halt if ψ is not of this form.

D2: Test if a, $\hat{a}$, and c belong to G; output reject and halt if this is not the case.

D3: Compute $v \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a}, c)$.

D4: Test if $d = a^{x_1+y_1 v}\hat{a}^{x_2+y_2 v}$; output reject and halt if this is not the case.

D5′: Compute $b \leftarrow a^z$.

D6: Compute $m \leftarrow c \cdot b^{-1}$ and output m.

FIG. 6.2. *Key generation and decryption algorithms for* CS1a.

Key Generation: On input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, compute

$$\Gamma[\hat{G}, G, g, q] \xleftarrow{R} \hat{S}(1^\lambda);\ \mathsf{hk} \xleftarrow{R} \mathsf{HF.KeySpace}_{\lambda,\Gamma};$$
$$w \xleftarrow{R} \mathbf{Z}_q^*;\ x, y, z \xleftarrow{R} \mathbf{Z}_q;$$
$$\hat{g} \leftarrow g^w;\ e \leftarrow g^x;\ f \leftarrow g^y;\ h \leftarrow g^z$$

and output the public key $\mathsf{PK} = (\Gamma, \mathsf{hk}, \hat{g}, e, f, h)$ and the secret key $\mathsf{SK} = (\Gamma, \mathsf{hk}, w, x, y, z)$.

Decryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a secret key

$$\mathsf{SK} = (\Gamma[\hat{G}, G, g, q], \mathsf{hk}, x, y, z) \in [\mathbf{S}_\lambda] \times [\mathsf{HF.KeySpace}_{\lambda,\Gamma}] \times \mathbf{Z}_q^3,$$

along with a ciphertext ψ, do the following.

D1: Parse ψ as a 4-tuple $(a, \hat{a}, c, d) \in \hat{G}^4$; output reject and halt if ψ is not of this form.

D2′: Test if a and c belong to G; output reject and halt if this is not the case.

D3: Compute $v \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a}, c)$.

D4′: Test if $\hat{a} = a^w$ and $d = a^{x+yv}$; output reject and halt if this is not the case.

D5′: Compute $b \leftarrow a^z$.

D6: Compute $m \leftarrow c \cdot b^{-1}$ and output m.

FIG. 6.3. *Key generation and decryption algorithms for* CS1b.

Our formal model of security does not model any notion of time at all, so such attacks fall outside of the model. While some cryptosystems are vulnerable to actual attacks given this type of timing information—notably, Manger's attack [41] on RSA

PKCS #1 version 2—we know of no actual timing attack along these lines on CS1b.

Remark 6.10. For the same reasons as discussed in the previous remark, it is important that any "error code" returned by the decryption algorithm in scheme CS1b not reveal the precise reason why a ciphertext was rejected. Again, we know of no actual "side channel" attack along these lines on CS1b.

THEOREM 6.10. *If the DDH assumption holds for $\mathcal{G}$ and the TCR assumption holds for* HF*, then* CS1a *and* CS1b *are secure against adaptive chosen ciphertext attack.*

In particular, for all probabilistic, polynomial-time oracle query machines A*, for all $\lambda \in \mathbf{Z}_{\geq 0}$, and for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we have*

$$|\mathsf{AdvCCA}_{\mathsf{CS1a},\mathsf{A}}(\lambda \mid \Gamma) - \mathsf{AdvCCA}_{\mathsf{CS1},\mathsf{A}}(\lambda \mid \Gamma)| \leq Q_{\mathsf{A}}(\lambda)/q \tag{6.11}$$

and

$$|\mathsf{AdvCCA}_{\mathsf{CS1b},\mathsf{A}}(\lambda \mid \Gamma) - \mathsf{AdvCCA}_{\mathsf{CS1},\mathsf{A}}(\lambda \mid \Gamma)| \leq Q_{\mathsf{A}}(\lambda)/q. \tag{6.12}$$

To prove this theorem, let us fix A, λ, and $\Gamma[\hat{G}, G, g, q]$. Consider the attack game $\mathbf{G}_0$ as defined in section 6.2: this is the game that A plays against the scheme CS1 for the given values of λ and Γ. We adopt all the notational conventions established at the beginning of section 6.2 (i.e., prior to the description of game $\mathbf{G}_1$).

We begin by defining two modifications of game $\mathbf{G}_0$.

Game $\mathbf{G}_{-1a}$. In this game, we modify the decryption oracle so that in place of step **D5** we execute step **D5′** as in the scheme CS1a. We emphasize that in game $\mathbf{G}_{-1a}$ we have $z = z_1 + z_2 w$, where w, z_1, and z_2 are generated by the key generation algorithm of CS1.

Game $\mathbf{G}_{-1b}$. In this game, we modify the decryption oracle so that in place of steps **D4** and **D5** we execute steps **D4′** and **D5′** as in the scheme CS1b. We emphasize that in game $\mathbf{G}_{-1b}$ we have $x = x_1 + x_2 w$, $y = x_1 + x_2 w$, and $z = z_1 + z_2 w$, where w, x_1, x_2, y_1, y_2, z_1, and z_2 are generated by the key generation algorithm of CS1.

Let T_{-1a} be the event that $\sigma = \hat{\sigma}$ in game $\mathbf{G}_{-1a}$ and T_{-1b} be the event that $\sigma = \hat{\sigma}$ in game $\mathbf{G}_{-1b}$.

We remind the reader that games $\mathbf{G}_0$, $\mathbf{G}_{-1a}$, and $\mathbf{G}_{-1b}$ all operate on the *same* underlying probability space: all of the variables

$$\mathsf{Coins}, \mathsf{hk}, w, x_1, x_2, y_1, y_2, z_1, z_2, \sigma, u^*$$

that ultimately determine the events T_0, T_{-1a}, and T_{-1b} have the same values in games $\mathbf{G}_0$, $\mathbf{G}_{-1a}$, and $\mathbf{G}_{-1b}$; all that changes is the functional behavior of the decryption oracle.

It is straightforward to verify that

$$\mathsf{AdvCCA}_{\mathsf{CS1a},\mathsf{A}}(\lambda \mid \Gamma) = |\Pr[T_{-1a} - 1/2]|$$

and

$$\mathsf{AdvCCA}_{\mathsf{CS1b},\mathsf{A}}(\lambda \mid \Gamma) = |\Pr[T_{-1b} - 1/2]|.$$

Let us define the event R_{-1b} to be the event that some ciphertext is rejected in game $\mathbf{G}_{-1b}$ in step **D4′** that would have passed the test in **D4**. It is clear that games $\mathbf{G}_0$, $\mathbf{G}_{-1a}$, and $\mathbf{G}_{-1b}$ all proceed identically until event R_{-1b} occurs. In particular, the

events $T_0 \wedge \neg R_{-1b}$, $T_{-1a} \wedge \neg R_{-1b}$, and $T_{-1b} \wedge \neg R_{-1b}$ are identical. So, by Lemma 6.2, we have

$$|\Pr[T_0] - \Pr[T_{-1a}]| \leq \Pr[R_{-1b}]$$

and

$$|\Pr[T_0] - \Pr[T_{-1b}]| \leq \Pr[R_{-1b}].$$

So it suffices to show that

$$\Pr[R_{-1b}] \leq Q_{\mathsf{A}}(\lambda)/q. \tag{6.13}$$

To do this, for $1 \leq i \leq Q_{\mathsf{A}}(\lambda)$, let $R^{(i)}_{-1b}$ be the event that there is an ith ciphertext submitted to the decryption oracle in game $\mathbf{G}_{-1b}$ and that this ciphertext is rejected in step **D4′** but would have passed the test in step **D4**.

The bound (6.13) will follow immediately from the following lemma.

LEMMA 6.11. *For all* $1 \leq i \leq Q_{\mathsf{A}}(\lambda)$*, we have* $\Pr[R^{(i)}_{-1b}] \leq 1/q$.

Proof. The proof of this is lemma is almost identical to that of Lemma 6.8. Note that in game $\mathbf{G}_{-1b}$ the encryption oracle uses the "real" encryption algorithm and so itself does not leak any additional information about (x_1, x_2, y_1, y_2). This is in contrast to game $\mathbf{G}_5$, where the encryption oracle does leak additional information.

Fix $1 \leq i \leq Q_{\mathsf{A}}(\lambda)$. Consider the quantities

$$X := (\mathsf{Coins}, \mathsf{hk}, w, z, \sigma, u^*)$$

and

$$X' := (x, y).$$

The values of X and X' completely determine the adversary's entire behavior in game $\mathbf{G}_5$, and hence determine if there is an ith decryption oracle query and, if so, the value of the corresponding ciphertext. Let us call X and X' *relevant* if for these values of X and X' there is an ith decryption oracle query and the corresponding ciphertext passes steps **D1** and **D2**.

It will suffice to prove that conditioned on any fixed, relevant values of X and X', the probability that $R^{(i)}_{-1b}$ occurs is bounded by $1/q$.

The remainder of the argument is *exactly* as in Lemma 6.8, except using X, X', and the notion of *relevant* as defined here. □

6.4. A hash-free variant. Our basic scheme CS1 requires a target collision resistant hash function. Qualitatively, the TCR assumption is much weaker than the DDH assumption, since one can build a target collision resistant hash function based on an arbitrary one-way function. Indeed, one can build a collision resistant hash function under the DL assumption; however, the hash functions arising from such a construction produce an output that is in G, whereas we need a hash function that maps into $\mathbf{Z}_q$. We cannot in general expect to find an easy-to-compute, injective map from G onto $\mathbf{Z}_q$; in Example 2 in section 4.2, we in fact do have such a map, but that is an exceptional case.

For these reasons, we present a variation CS2 of our basic scheme that does not require a hash function.

Key Generation: On input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, compute

$$\begin{array}{l}\Gamma[\hat{G}, G, g, q] \xleftarrow{R} \hat{S}(1^\lambda);\\ w \xleftarrow{R} \mathbf{Z}_q^*;\ x_1, x_2, z_1, z_2 \xleftarrow{R} \mathbf{Z}_q;\\ \text{for } i = 1, \ldots, N\text{: } y_1^{(i)}, y_2^{(i)} \xleftarrow{R} \mathbf{Z}_q;\\ \hat{g} \leftarrow g^w;\ e \leftarrow g^{x_1}\hat{g}^{x_2};\ h \leftarrow g^{z_1}\hat{g}^{z_2};\\ \text{for } i = 1, \ldots, N\text{: } f_i \leftarrow g^{y_1^{(i)}}\hat{g}^{y_2^{(i)}}\end{array}$$

and output the public key $\mathsf{PK} = (\Gamma, \hat{g}, e, (f_i)_{i=1}^N, h)$ and the secret key $\mathsf{SK} = (\Gamma, x_1, x_2, (y_1^{(i)}, y_2^{(i)})_{i=1}^N, z_1, z_2)$.

Encryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a public key

$$\mathsf{PK} = (\Gamma[\hat{G}, G, g, q], \hat{g}, e, (f_i)_{i=1}^N, h) \in [\mathbf{S}_\lambda] \times G^{N+3},$$

along with a message $m \in G$, compute

E1: $u \xleftarrow{R} \mathbf{Z}_q$;
E2: $a \leftarrow g^u$;
E3: $\hat{a} \leftarrow \hat{g}^u$;
E4: $b \leftarrow h^u$;
E5: $c \leftarrow b \cdot m$;
E6: $(v_1, \ldots, v_N) \leftarrow \mathsf{Chop}_{\lambda,\Gamma}(a, \hat{a}, c)$;
E7: $d \leftarrow e^u \prod_{i=1}^N f_i^{uv_i}$

and output the ciphertext $\psi = (a, \hat{a}, c, d)$.

Decryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a secret key

$$\mathsf{SK} = (\Gamma[\hat{G}, G, g, q], x_1, x_2, (y_1^{(i)}, y_2^{(i)})_{i=1}^N, z_1, z_2) \in [\mathbf{S}_\lambda] \times \mathbf{Z}_q^{N+4},$$

along with a ciphertext ψ, do the following.

D1: Parse ψ as a 4-tuple $(a, \hat{a}, c, d) \in \hat{G}^4$; output reject and halt if ψ is not of this form.
D2: Test if a, $\hat{a}$, and c belong to G; output reject and halt if this is not the case.
D3: Compute $(v_1, \ldots, v_N) \leftarrow \mathsf{Chop}_{\lambda,\Gamma}(a, \hat{a}, c)$.
D4: Test if $d = a^{x_1 + \sum_{i=1}^N y_1^{(i)} v_i} \cdot \hat{a}^{x_2 + \sum_{i=1}^N y_2^{(i)} v_i}$; output reject and halt if this is not the case.
D5: Compute $b \leftarrow a^{z_1}\hat{a}^{z_2}$.
D6: Compute $m \leftarrow c \cdot b^{-1}$ and output m.

FIG. 6.4. *The public-key encryption scheme* CS2*, where* $N = N(\lambda, \Gamma)$.

This scheme requires a family $\{\mathsf{Chop}_{\lambda,\Gamma}\}$ of "chopping" functions associated with the group scheme $\mathcal{G}$ with the following properties. For $\lambda \in \mathbf{Z}_{\geq 0}$ and $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, the function $\mathsf{Chop}_{\lambda,\Gamma}$ *injectively* maps triples $\rho \in G^3$ of group elements to N-tuples $(v_1, \ldots, v_N) \in \mathbf{Z}_q^N$. Here, $N = N(\lambda, \Gamma)$ is bounded by a polynomial in λ, and the function $\mathsf{Chop}_{\lambda,\Gamma}$ should be computable by a deterministic, polynomial-time function that takes inputs 1^λ, Γ, and ρ.

In principle, such chopping functions always exist, since we can write down the binary representation of ρ and chop it into bit strings of length $\lfloor \log_2 q \rfloor$.

We present the details of scheme CS2 in Figure 6.4.

THEOREM 6.12. *If the DDH assumption holds for $\mathcal{G}$, then* CS2 *is secure against adaptive chosen ciphertext attack.*

In particular, for all probabilistic, polynomial-time oracle query machines A*, there exists a probabilistic algorithm* A_1*, whose running time is essentially the same as that of* A*, such that the following holds. For all $\lambda \in \mathbf{Z}_{\geq 0}$ and all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we have*

$$\mathsf{AdvCCA}_{\mathsf{CS2},\mathsf{A}}(\lambda \mid \Gamma) \leq \mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}_1}(\lambda \mid \Gamma) + (Q_{\mathsf{A}}(\lambda) + 3)/q.$$

The proof of this theorem follows along the same lines as the proof of Theorem 6.1. We present here a sketch of the proof, appealing in several places to arguments found in the proof of Theorem 6.1 so as to avoid repeating arguments that are identical or nearly identical.

Let us fix a probabilistic, polynomial-time oracle query machine A, the value of the security parameter $\lambda \in \mathbf{Z}_{\geq 0}$, and the group description $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$.

We define $x, z \in \mathbf{Z}_q$ as follows:

$$x := x_1 + x_2 w, \;\; z := z_1 + z_2 w.$$

We also define $y^{(i)} \in \mathbf{Z}_q$, for $1 \leq i \leq N$, as

$$y^{(i)} := y_1^{(i)} + y_2^{(i)} w.$$

As a notational convention, whenever a particular ciphertext ψ is under consideration in some context, the following values are also implicitly defined in that context:

- $a, \hat{a}, c, d \in G$, where $\psi = (a, \hat{a}, c, d)$;
- $u, \hat{u}, v_1, \ldots, v_N, r, s \in \mathbf{Z}_q$, where

$$u := \log_g a, \;\; \hat{u} := \log_{\hat{g}} \hat{a}, \;\; (v_1, \ldots, v_N) := \mathsf{Chop}_{\lambda,\Gamma}(a, \hat{a}, c),$$
$$r := \log_g c, \;\; s := \log_g d.$$

For the target ciphertext ψ^*, we also denote by $a^*, \hat{a}^*, c^*, d^* \in G$ and $u^*, \hat{u}^*, v_1^*, \ldots, v_N^*, r^*, s^* \in \mathbf{Z}_q$ the corresponding values.

The probability space defining the attack game is then determined by the following, mutually independent, random variables:

- the coin tosses of A;
- the values $w, x_1, x_2, y_1^{(1)}, \ldots, y^{(N)}, y_2^{(1)}, \ldots, y_2^{(N)}, z_1, z_2$ generated by the key generation algorithm;
- the values $\sigma \in \{0, 1\}$ and $u^* \in \mathbf{Z}_q$ generated by the encryption oracle.

Let $\mathbf{G}_0$ be the original attack game, let $\hat{\sigma} \in \{0, 1\}$ denote the output of A, and let T_0 be the event that $\sigma = \hat{\sigma}$ in $\mathbf{G}_0$ so that $\mathsf{AdvCCA}_{\mathsf{CS2},\mathsf{A}}(\lambda \mid \Gamma) = |\Pr[T_0] - 1/2|$.

As in the proof of Theorem 6.1, we shall define a sequence of modified games $\mathbf{G}_i$, for $i = 1, 2, \ldots$, and in game $\mathbf{G}_i$ the event T_i will be the event corresponding to event T_0 but in game $\mathbf{G}_i$. We remind the reader that all of these games operate on the same underlying probability space, and, except as otherwise specified, random variables have identical values between games.

Game $\mathbf{G}_1$. In game $\mathbf{G}_1$, we modify the algorithm used by the encryption oracle as follows. Steps **E4** and **E7** are replaced by

E4′: $b \leftarrow a^{z_1} \hat{a}^{z_2}$;

E7′: $d \leftarrow a^{x_1 + \sum_{i=1}^N y_1^{(i)} v_i} \cdot \hat{a}^{x_2 + \sum_{i=1}^N y_2^{(i)} v_i}$.

By the same reasoning as in the proof of Theorem 6.1, we have $\Pr[T_1] = \Pr[T_0]$.

Game $\mathbf{G_2}$. We again modify the encryption oracle, replacing step **E3** by

E3′: $\hat{u} \xleftarrow{R} \mathbf{Z}_q \setminus \{u\}$; $\hat{a} \leftarrow \hat{g}^{\hat{u}}$.

By the same reasoning as in the proof of Theorem 6.1, one sees that there exists a probabilistic algorithm A_1, whose running time is essentially the same as that of A, such that

$$|\Pr[T_2] - \Pr[T_1]| \leq \mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}_1}(\lambda \mid \Gamma) + 3/q.$$

Game $\mathbf{G_3}$. In this game, we modify the decryption oracle in game $\mathbf{G}_2$, replacing steps **D4** and **D5** by

D4′: test if $\hat{a} = a^w$ and $d = a^{x+\sum_{i=1}^{N} y^{(i)} v_i}$; output reject and halt if this is not the case;

D5′: $b \leftarrow a^z$.

Let R_3 be the event that in game $\mathbf{G}_3$ some ciphertext ψ is submitted to the decryption oracle that is rejected in step **D4′** but that would have passed the test in step **D4**.

As in the proof of Theorem 6.1, we have

$$|\Pr[T_3] - \Pr[T_2]| \leq \Pr[R_3].$$

We claim that

$$\Pr[R_3] \leq Q_{\mathsf{A}}(\lambda)/q.$$

We can prove the analogue of Lemma 6.6 (in game $\mathbf{G}_5$ in the proof of Theorem 6.1) by considering an $(N+3) \times (2N+2)$ matrix M over $\mathbf{Z}_q$ defined as

$$M := \begin{pmatrix} 1 & w & & & & & & \\ & & 1 & w & & & & \\ & & & & \ddots & & & \\ & & & & & 1 & w & \\ u^* & \hat{u}^* w & u^* v_1^* & \hat{u}^* v_1^* w & \cdots & u^* v_N^* & \hat{u}^* v_N^* w \\ u & \hat{u} w & u v_1 & \hat{u} v_1 w & \cdots & u v_N & \hat{u} v_N w \end{pmatrix},$$

where $w \neq 0$, $\hat{u} \neq u$, $\hat{u}^* \neq u^*$, and $v_i \neq v_i^*$ for some $i \in \{1, \ldots, N\}$. It will suffice to show that the rows of M are linearly independent.

If we choose i such that $v_i \neq v_i^*$ and consider the 4×4 submatrix M' of M consisting of the intersection of columns 1, 2, $2i+1$, $2i+2$ of M and rows 1, $i+1$, $N+2$, $N+3$ of M, we see that matrix M' has the same form as the matrix considered in Lemma 6.9, and hence is nonsingular. It follows that the rows of M are linearly independent, since any nontrivial linear relation among the rows of M implies a nontrivial linear relation among the rows of M'.

Game $\mathbf{G_4}$. We again modify the algorithm used by the encryption oracle, replacing step **E5** by

E5′: $r \xleftarrow{R} \mathbf{Z}_q$; $c \leftarrow g^r$.

By reasoning analogous to that in game $\mathbf{G}_4$ in the proof of Theorem 6.1, one can show that

$$\Pr[T_4] = \Pr[T_3].$$

Moreover, by construction it is evident that

$$\Pr[T_4] = 1/2.$$

That completes the proof sketch of Theorem 6.12. We leave it to the reader to work out the details of the design and analysis of variants CS2a and CS2b of scheme CS2, corresponding to the variants CS1a and CS1b of scheme CS1, which were discussed in section 6.3.

Remark 6.11. Note that the high-level structure of the proof of Theorem 6.12 is significantly simpler than that of Theorem 6.1. In particular, in the analysis of game $\mathbf{G}_3$ in the proof of Theorem 6.12 we were able to bound the quantity $\Pr[R_3]$ directly without deferring the analysis to a later game, as in the proof of Theorem 6.1. This simplification comes from the fact that we do not have to deal with a target collision resistant hash function in Theorem 6.12, as we did in Theorem 6.1. Indeed, if in the scheme CS1 we use a collision resistant hash function, we could prove the security of CS1 using a proof with essentially the same line of reasoning as that of the proof of Theorem 6.12, with one extra game between $\mathbf{G}_0$ and $\mathbf{G}_1$ to effectively ban hash function collisions.

7. Hybrid encryption. The encryption schemes presented in the previous section all had restricted message spaces. In some settings, an encryption scheme with an unrestricted message space is more desirable. A simple and efficient way to build an encryption scheme that has an unrestricted message is to build a *hybrid* encryption scheme. Loosely speaking, such a scheme uses public-key encryption techniques to encrypt a key K that is then used to encrypt the actual message using symmetric-key encryption techniques. In this section, we develop the necessary tools for building a hybrid public-key encryption scheme.

One key ingredient in any hybrid scheme is a *key encapsulation mechanism*. This is like a public-key encryption scheme, except that the job of the encryption algorithm is to generate the encryption of a random key K. Of course, one can always use a general-purpose public-key encryption scheme to do this by simply generating K at random and then encrypting it. However, there are typically more efficient ways to do this.

As a quick example of a key encapsulation mechanism, consider the following variation of the ElGamal encryption scheme. Let G be a group of prime order q generated by an element g. Let H be a cryptographic hash function, such as SHA-1. The public key consists of a group element $h = g^z$, where $z \in \mathbf{Z}_q$ is chosen at random; the secret key is z. To generate an encryption of a symmetric key K, we compute

$$u \xleftarrow{R} \mathbf{Z}_q;\ a \leftarrow g^u;\ b \leftarrow h^u;\ K \leftarrow H(b)$$

to form a ciphertext $\psi = a$. To decrypt a ciphertext $\psi = a$ using the secret key, one computes

$$b \leftarrow a^z;\ K \leftarrow H(b),$$

obtaining a symmetric key K.

To build a complete hybrid encryption scheme, we combine a key encapsulation mechanism with a symmetric-key encryption scheme.

7.1. Key encapsulation. A key encapsulation mechanism KEM consists of the following algorithms:

- A probabilistic, polynomial-time *key generation algorithm* $\mathsf{KEM.KeyGen}$ that on input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$ outputs a public-key/secret-key pair $(\mathsf{PK}, \mathsf{SK})$. The structure of PK and SK depends on the particular scheme.
  For $\lambda \in \mathbf{Z}_{\geq 0}$, we define the probability spaces

$$\mathsf{KEM.PKSpace}_\lambda := \{\mathsf{PK} : (\mathsf{PK}, \mathsf{SK}) \xleftarrow{R} \mathsf{KEM.KeyGen}(1^\lambda)\}$$

  and

$$\mathsf{KEM.SKSpace}_\lambda := \{\mathsf{SK} : (\mathsf{PK}, \mathsf{SK}) \xleftarrow{R} \mathsf{KEM.KeyGen}(1^\lambda)\}.$$

- A probabilistic, polynomial-time encryption algorithm $\mathsf{KEM.Encrypt}$ that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$ and a public key $\mathsf{PK} \in [\mathsf{KEM.PKSpace}_\lambda]$ and outputs a pair (K, ψ), where K is a *key* and ψ is a ciphertext.
  A key K is a bit string of length $\mathsf{KEM.KeyLen}(\lambda)$, where $\mathsf{KEM.KeyLen}(\lambda)$ is another parameter of the key encapsulation mechanism.
  A ciphertext is a bit string.
- A deterministic, polynomial-time decryption algorithm $\mathsf{KEM.Decrypt}$ that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a secret key $\mathsf{SK} \in [\mathsf{KEM.SKSpace}_\lambda]$, and a ciphertext ψ and outputs either a key K or the special symbol reject.

7.1.1. Soundness. As for public key encryption, we need an appropriate notion of soundness. A definition of soundness that is adequate for our purposes runs as follows. Let us say a public-key/secret-key pair $(\mathsf{PK}, \mathsf{SK}) \in [\mathsf{KEM.KeyGen}(1^\lambda)]$ is *bad* if for some $(K, \psi) \in [\mathsf{KEM.Encrypt}(1^\lambda, \mathsf{PK})]$ we have $\mathsf{KEM.Decrypt}(1^\lambda, \mathsf{SK}, \psi) \neq K$. Let $\mathsf{BadKeyPair}_{\mathsf{KEM}}(\lambda)$ denote the probability that the key generation algorithm generates a bad key pair for a given value of λ. Then our requirement is that $\mathsf{BadKeyPair}_{\mathsf{KEM}}(\lambda)$ grows negligibly in λ.

7.1.2. Security against adaptive chosen ciphertext attack. For a key encapsulation mechanism, an adversary A in an adaptive chosen ciphertext attack is a probabilistic, polynomial-time oracle query machine that takes as input 1^λ, where $\lambda \in \mathbf{Z}_{\geq 0}$ is the security parameter. We now describe the attack game used to define security against adaptive chosen ciphertext attack, which is quite similar to that used to define the corresponding notion of security for a public-key encryption scheme.

Stage 1: The adversary queries a *key generation oracle*. The key generation oracle computes $(\mathsf{PK}, \mathsf{SK}) \xleftarrow{R} \mathsf{KEM.KeyGen}(1^\lambda)$ and responds with PK.

Stage 2: The adversary makes a sequence of calls to a *decryption oracle*.

For each decryption oracle query, the adversary submits a ciphertext ψ, and the decryption oracle responds with $\mathsf{KEM.Decrypt}(1^\lambda, \mathsf{SK}, \psi)$.

Stage 3: The adversary queries an *encryption oracle*.

The encryption oracle computes

$$(K^*, \psi^*) \xleftarrow{R} \mathsf{KEM.Encrypt}(1^\lambda, \mathsf{PK});\ K^+ \xleftarrow{R} \{0,1\}^\ell;\ \tau \xleftarrow{R} \{0,1\};$$
$$\text{if } \tau = 0, \text{ then } K^\dagger \leftarrow K^*; \text{ else } K^\dagger \leftarrow K^+,$$

where $\ell := \mathsf{KEM.KeyLen}(\lambda)$, and responds with the pair $(K^\dagger, \psi^*)$.

Stage 4: The adversary continues to make calls to the decryption oracle subject only to the restriction that a submitted ciphertext ψ is not *identical* to ψ^*.

Stage 5: The adversary outputs $\hat{\tau} \in \{0,1\}$.

We define $\mathsf{AdvCCA}_{\mathsf{KEM},\mathsf{A}}(\lambda)$ to be $|\Pr[\tau = \hat{\tau}] - 1/2|$ in the above attack game.

We say that KEM is *secure against adaptive chosen ciphertext attack* if
for all probabilistic, polynomial-time oracle query machines A *the function* $\mathsf{AdvCCA}_{\mathsf{KEM},\mathsf{A}}(\lambda)$ *grows negligibly in* λ.

In applying the above definition of security, one typically works directly with the quantity

$$\mathsf{AdvCCA}'_{\mathsf{KEM},\mathsf{A}}(\lambda) := |\Pr[\hat{\tau} = 1 \mid \tau = 0] - \Pr[\hat{\tau} = 1 \mid \tau = 1]|\,.$$

It is easy to verify that

$$\mathsf{AdvCCA}'_{\mathsf{KEM},\mathsf{A}}(\lambda) = 2 \cdot \mathsf{AdvCCA}_{\mathsf{KEM},\mathsf{A}}(\lambda).$$

7.2. One-time symmetric-key encryption. A *one-time symmetric-key encryption scheme* SKE consists of two algorithms:

- A deterministic, polynomial-time encryption algorithm $\mathsf{SKE.Encrypt}$ that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a key K, and a message m and outputs a ciphertext χ.
  The key K is a bit string of length $\mathsf{SKE.KeyLen}(\lambda)$.
  Here, $\mathsf{SKE.KeyLen}(\lambda)$ is a parameter of the encryption scheme, which we assume can be computed in deterministic polynomial time given 1^λ.
  The message m is a bit string of arbitrary, unbounded length.
  The ciphertext χ is a bit string.
  We denote by $\mathsf{SKE.CTLen}(\lambda, \ell)$ the maximum length of any encryption of a message of length at most ℓ when using security parameter λ.
- A deterministic, polynomial-time decryption algorithm $\mathsf{SKE.Decrypt}$ that takes as input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a key K, and a ciphertext χ and outputs a message m or the special symbol reject.
  The key K is a bit string of length $\mathsf{SKE.KeyLen}(\lambda)$.
  The ciphertext χ is a bit string of arbitrary length.

We require that SKE satisfy the following *soundness* condition: for all $\lambda \in \mathbf{Z}_{\geq 0}$, for all $K \in \{0,1\}^{\mathsf{SKE.KeyLen}(\lambda)}$, for all $m \in \{0,1\}^*$, we have

$$\mathsf{SKE.Decrypt}(1^\lambda, K, \mathsf{SKE.Encrypt}(1^\lambda, K, m)) = m.$$

7.2.1. Two definitions of security. We define two notions of security for a one-time symmetric-key encryption scheme: security against passive attacks and security against adaptive chosen ciphertext attacks.

As usual, an adversary A is a probabilistic, polynomial-time oracle query machine that takes as input 1^λ, where $\lambda \in \mathbf{Z}_{\geq 0}$ is the security parameter.

A *passive* attack runs as follows. The adversary A chooses two messages, m_0 and m_1, of equal length and gives these to an *encryption oracle*. The encryption oracle generates a random key K of length $\mathsf{SKE.KeyLen}(\lambda)$, along with random $\sigma \in \{0,1\}$, and encrypts the message m_σ using the key K. The adversary A is then given the resulting ciphertext χ^*. Finally, the adversary outputs $\hat{\sigma} \in \{0,1\}$.

We define $\mathsf{AdvPA}_{\mathsf{SKE},\mathsf{A}}(\lambda)$ to be $|\Pr[\sigma = \hat{\sigma}] - 1/2|$ in the above attack game.

We say that SKE is *secure against passive attacks* if
for all probabilistic, polynomial-time oracle query machines A *the function* $\mathsf{AdvPA}_{\mathsf{SKE},\mathsf{A}}(\lambda)$ *grows negligibly in* λ.

An *adaptive chosen ciphertext attack* is exactly the same as a passive attack, except that *after* the adversary A obtains the target ciphertext χ^* from the encryption oracle the adversary may then query a *decryption oracle* any number of times. In each

decryption oracle query, A submits a ciphertext $\chi \neq \chi^*$ and obtains the decryption of χ under the key K. As in the passive attack, A outputs $\hat{\sigma} \in \{0,1\}$.

We define $\mathsf{AdvCCA}_{\mathsf{SKE},\mathsf{A}}(\lambda)$ to be $|\Pr[\sigma = \hat{\sigma}] - 1/2|$ in the above attack game.

We say that SKE is *secure against adaptive chosen ciphertext attacks* if *for all probabilistic, polynomial-time oracle query machines* A *the function* $\mathsf{AdvCCA}_{\mathsf{SKE},\mathsf{A}}(\lambda)$ *grows negligibly in* λ.

7.2.2. Constructions. Our definition of a symmetric-key encryption scheme and the corresponding notions of security are tailored to the application of building a hybrid public-key encryption scheme. These definitions may not be appropriate for other settings. In particular, our definitions of security do not imply protection against chosen plaintext attack; however, this protection is not needed for hybrid public-key encryption schemes, since a symmetric key is only used to encrypt a single message.

It is easy to build a symmetric key encryption scheme that achieves security against passive attacks using standard symmetric-key techniques. For example, to encrypt a message m, one can expand the key K using a pseudorandom bit generator to obtain a "one time pad" α of length $|m|$ and then compute $\chi \leftarrow m \oplus \alpha$.

A pseudorandom bit generator can be built from an arbitrary one-way permutation [33] or even from an arbitrary one-way function [36, 35]. These constructions, however, are not very practical. In a practical implementation, it is perfectly reasonable to stretch the key K by using it as the key to a dedicated block cipher and then evaluate the block cipher at successive points (so-called counter mode) to obtain a sequence of pseudorandom bits (cf. [44, Chapter 7]).

Note that the above construction yields a scheme that is completely insecure against adaptive chosen ciphertext attack. However, it is also easy to build a symmetric key encryption scheme $\mathsf{SKE2}$ that achieves security against adaptive chosen ciphertext attack, given an arbitrary scheme $\mathsf{SKE1}$ that is only secure against passive attacks.

One technique is to simply build an $\mathsf{SKE2}$ ciphertext by attaching a *message authentication code* to the $\mathsf{SKE1}$ ciphertext. Although this technique seems to be "folklore," for completeness we develop the details here.

A *one-time message authentication code* MAC specifies the following items:

- For $\lambda \in \mathbf{Z}_{\geq 0}$, a *key length* parameter $\mathsf{MAC.KeyLen}(\lambda)$ and an *output length* parameter $\mathsf{MAC.OutLen}(\lambda)$.
  We assume that $\mathsf{MAC.KeyLen}(\lambda)$ can be computed in deterministic polynomial time given 1^λ.
- A family of functions indexed by $\lambda \in \mathbf{Z}_{\geq 0}$ and $\mathsf{mk} \in \{0,1\}^{\mathsf{MAC.KeyLen}(\lambda)}$, where each function $\mathsf{MAC}^\lambda_{\mathsf{mk}}$ maps arbitrary bit strings to bit strings of length exactly $\mathsf{MAC.OutLen}(\lambda)$.
  There must be a deterministic, polynomial-time algorithm that on input 1^λ, $\mathsf{mk} \in \{0,1\}^{\mathsf{MAC.KeyLen}(\lambda)}$, and $\alpha \in \{0,1\}^*$ outputs $\mathsf{MAC}^\lambda_{\mathsf{mk}}(\alpha)$.

To define security for MAC, we define an attack game as follows. As usual, an adversary A is a probabilistic, polynomial-time oracle query machine that takes as input 1^λ, where $\lambda \in \mathbf{Z}_{\geq 0}$ is the security parameter. The adversary A first chooses a bit string α and submits this to an oracle. The oracle generates a random key mk of length $\mathsf{MAC.KeyLen}(\lambda)$, computes $\beta \leftarrow \mathsf{MAC}^\lambda_{\mathsf{mk}}(\alpha)$, and returns β to the adversary. The adversary A then outputs a list

$$((\alpha_1, \beta_1), \ldots, (\alpha_k, \beta_k))$$

of pairs of bit strings. We say that A has *produced a forgery* if for some $1 \le i \le k$ we have $\alpha_i \ne \alpha$ and $\mathsf{MAC}^{\lambda}_{\mathsf{mk}}(\alpha_i) = \beta_i$.

We say that A is a $(L_1(\lambda), L_2(\lambda), N(\lambda))$ forging adversary if $|\alpha| \le L_1(\lambda)$, $k \le N(\lambda)$, and $|\alpha_i| \le L_2(\lambda)$ for all $1 \le i \le k$.

Define $\mathsf{AdvForge}_{\mathsf{MAC},\mathsf{A}}(\lambda)$ to be the probability that A produces a forgery in the above game. We say that MAC is secure if

> *for all probabilistic, polynomial-time oracle query machines* A *the function* $\mathsf{AdvForge}_{\mathsf{MAC},\mathsf{A}}(\lambda)$ *grows negligibly in* λ.

Message authentication codes have been extensively studied (cf. [44, Chapter 9]). Once can easily build secure one-time message authentication codes using an appropriate family of universal hash functions without relying on any intractability assumptions. There are also other ways to build message authentication codes which may be preferable in practice, even though the security of these schemes is not fully proven.

Now we demonstrate how to use $\mathsf{SKE1}$ and MAC to build $\mathsf{SKE2}$. The key length $\mathsf{SKE2.KeyLen}(\lambda)$ of $\mathsf{SKE2}$ will be equal to

$$\mathsf{SKE1.KeyLen}(\lambda) + \mathsf{MAC.KeyLen}(\lambda).$$

We will write such a key as (K, mk), where K is a bit string of length $\mathsf{SKE1.KeyLen}(\lambda)$ and mk is a bit string of length $\mathsf{MAC.KeyLen}(\lambda)$.

To encrypt a message m under a key (K, mk) as above, algorithm $\mathsf{SKE2.Encrypt}$ computes

$$\chi \leftarrow \mathsf{SKE1.Encrypt}(1^{\lambda}, K, m);\ \mathsf{tag} \leftarrow \mathsf{MAC}^{\lambda}_{\mathsf{mk}}(\chi);\ \chi' \leftarrow \chi \,\|\, \mathsf{tag}$$

and outputs the ciphertext χ'.

To decrypt a ciphertext χ' under a key (K, mk) as above, algorithm $\mathsf{SKE2.Decrypt}$ first parses χ' as $\chi' = \chi \,\|\, \mathsf{tag}$, where tag is a bit string of length $\mathsf{MAC.OutLen}(\lambda)$. If this parsing step fails (because χ' is too short), then the algorithm outputs reject; otherwise, it computes

$$\mathsf{tag}' \leftarrow \mathsf{MAC}^{\lambda}_{\mathsf{mk}}(\chi).$$

If $\mathsf{tag} \ne \mathsf{tag}'$, the algorithm outputs reject; otherwise, it computes

$$m \leftarrow \mathsf{SKE1.Decrypt}(1^{\lambda}, K, \chi)$$

and outputs m.

To analyze the security of $\mathsf{SKE2}$, we recall that for all probabilistic, polynomial-time oracle query machines A, for all $\lambda \in \mathbf{Z}_{\ge 0}$, we denote by $Q_{\mathsf{A}}(\lambda)$ an upper bound on the number of decryption oracle queries made by A on input 1^{λ}. Although we introduced this notation in the context of public-key encryption, we can adopt it here in the context of symmetric-key encryption as well. We remind the reader that $Q_{\mathsf{A}}(\lambda)$ should be a strict bound that holds for any environment.

For all probabilistic, polynomial-time oracle query machines A, for all $\lambda \in \mathbf{Z}_{\ge 0}$, we define $B_{\mathsf{A}}(\lambda)$ to be an upper bound on the length of the messages submitted by A to the encryption oracle and $B'_{\mathsf{A}}(\lambda)$ to be an upper bound on the ciphertexts submitted by A to the decryption oracle. As usual, these upper bounds should hold regardless of the environment of A.

THEOREM 7.1. *If* $\mathsf{SKE1}$ *is secure against passive attacks and* MAC *is a secure one-time message authentication code, then* $\mathsf{SKE2}$ *is secure against adaptive chosen ciphertext attacks.*

In particular, for every probabilistic, polynomial-time oracle query machine A*, there exist probabilistic oracle query machines* A_1 *and* A_2*, whose running times are essentially the same as that of* A*, such that for all* $\lambda \in \mathbf{Z}_{\geq 0}$*,*

$$\mathsf{AdvCCA}_{\mathsf{SKE2},\mathsf{A}}(\lambda) \leq \mathsf{AdvPA}_{\mathsf{SKE1},\mathsf{A}_1}(\lambda) + \mathsf{AdvForge}_{\mathsf{MAC},\mathsf{A}_2}(\lambda).$$

Moreover, A_2 *is a*

$$(\mathsf{SKE1.CTLen}(\lambda, B(\lambda)),\ B'(\lambda) - \mathsf{MAC.OutLen}(\lambda),\ Q_{\mathsf{A}}(\lambda))$$

forging adversary.

Proof. Fix A and λ, and let $\mathbf{G}_0$ denote the original chosen ciphertext attack game. Let T_0 be the event that $\sigma = \hat{\sigma}$ in game $\mathbf{G}_0$.

We next define a modified attack game $\mathbf{G}_1$ in which all ciphertexts submitted to the decryption oracle by A in game $\mathbf{G}_1$ are simply rejected.

Let T_1 be the event that $\sigma = \hat{\sigma}$ in game $\mathbf{G}_1$. Let R_1 be the event in game $\mathbf{G}_1$ that some ciphertext is rejected in game $\mathbf{G}_1$ that would not have been rejected under the rules of game $\mathbf{G}_0$.

Since games $\mathbf{G}_0$ and $\mathbf{G}_1$ proceed identically until event R_1 occurs, the events $T_0 \wedge \neg R_1$ and $T_1 \wedge \neg R_1$ are identical, and so, by Lemma 6.2, we have $|\Pr[T_0] - \Pr[T_1]| \leq \Pr[R_1]$.

It is straightforward to verify that

$$\Pr[R_1] \leq \mathsf{AdvForge}_{\mathsf{MAC},\mathsf{A}_2}(\lambda) \tag{7.1}$$

for an adversary A_2 as described above.

The theorem now follows by observing that the attack by A in game $\mathbf{G}_1$ is now a passive attack. That is, the adversary A_1 in the theorem simply runs the adversary A, and whenever A makes a decryption oracle query adversary A_1 simply lets A continue *as if* the decryption oracle rejected the ciphertext. □

Remark 7.1. Although the keys for SKE2 are longer than those for SKE1, this need not be the case if we use a pseudorandom bit generator to stretch a short key into a suitably long key. Indeed, the key length of any symmetric key encryption scheme need be no longer than the key length of a secure pseudorandom bit generator.

7.3. A hybrid construction. Let KEM be a key encapsulation mechanism (as defined in section 7.1), and let SKE be a one-time symmetric key encryption scheme (as defined in section 7.2). Further, let us assume that the two schemes are *compatible* in the sense that for all $\lambda \in \mathbf{Z}_{\geq 0}$ we have $\mathsf{KEM.KeyLen}(\lambda) = \mathsf{SKE.KeyLen}(\lambda)$. We now describe a hybrid public-key encryption scheme HPKE.

The key generation algorithm for HPKE is the same as that of KEM, and the public and secret keys are the same as those of KEM.

To encrypt a message m in the hybrid scheme, we run KEM.Encrypt to generate a symmetric key K and a ciphertext ψ encrypting K. We then encrypt m under the key K using SKE.Encrypt, obtaining a ciphertext χ. The output of the encryption algorithm is $\hat{\psi} = (\psi, \chi)$, encoded in a canonical fashion as a bit string.

The decryption algorithm for the hybrid scheme runs as follows. Given a ciphertext $\hat{\psi}$, we first verify that $\hat{\psi}$ properly encodes a pair (ψ, χ). If not, we output reject and halt. Next, we decrypt ψ using KEM.Decrypt; if this yields reject, then we output reject and halt. Otherwise, we obtain a symmetric key K and decrypt χ under K using SKE.Decrypt and output the resulting decryption (which may be reject).

THEOREM 7.2. *If* KEM *and* SKE *are secure against adaptive chosen ciphertext attacks, then so is* HPKE.

In particular, if A *is a probabilistic, polynomial-time oracle query machine, then there exist probabilistic oracle query machines* A_1 *and* A_2*, whose running times are essentially the same as that of* A*, such that for all* $\lambda \in \mathbf{Z}_{\geq 0}$ *we have*

$$\mathsf{AdvCCA}_{\mathsf{HPKE},\mathsf{A}}(\lambda) \leq \mathsf{BadKeyPair}_{\mathsf{KEM}}(\lambda) + \mathsf{AdvCCA}'_{\mathsf{KEM},\mathsf{A}_1}(\lambda) + \mathsf{AdvCCA}_{\mathsf{SKE},\mathsf{A}_2}(\lambda).$$

Proof. Fix A and λ, and let $\mathbf{G}_0$ be the original chosen ciphertext attack game played by A against HPKE. We let $\hat{\psi}^* = (\psi^*, \chi^*)$ denote the target ciphertext; σ is the hidden bit generated by the encryption oracle and $\hat{\sigma}$ is the bit output by A. Let T_0 be the event that $\sigma = \hat{\sigma}$. Also, let K^* denote the symmetric key output by the algorithm KEM.Encrypt during the encryption process within the encryption oracle.

We now define a modified game $\mathbf{G}_1$. In this game, whenever a ciphertext (ψ, χ) is submitted to the decryption oracle after the invocation of the encryption oracle, if $\psi = \psi^*$ (but $\chi \neq \chi^*$ of course), then the decryption oracle does not apply algorithm KEM.Decrypt to obtain the symmetric key but instead just uses the key K^* produced by the encryption oracle. Let T_1 be the event that $\sigma = \hat{\sigma}$ in game $\mathbf{G}_1$.

This change is slightly more than just conceptual, since KEM.KeyGen may generate a bad key pair with probability $\mathsf{BadKeyPair}_{\mathsf{KEM}}(\lambda)$. However, unless this occurs, games $\mathbf{G}_0$ and $\mathbf{G}_1$ proceed identically, and so, by Lemma 6.2, we have

$$|\Pr[T_1] - \Pr[T_0]| \leq \mathsf{BadKeyPair}_{\mathsf{KEM}}(\lambda).$$

Now we define a modified game $\mathbf{G}_2$. This game behaves just like game $\mathbf{G}_1$, except that we use a completely random symmetric key K^+ in place of the key K^* in both the encryption and decryption oracles. Let T_2 be the event that $\sigma = \hat{\sigma}$ in game $\mathbf{G}_2$.

It is straightforward to see that there is an oracle query machine A_1, whose running time is essentially the same as that of A, such that

$$|\Pr[T_2] - \Pr[T_1]| = \mathsf{AdvCCA}'_{\mathsf{KEM},\mathsf{A}_1}(\lambda).$$

The adversary A_1 basically just runs the adversary A. In the attack game that A_1 is playing against KEM, the value $K^\dagger$ is equal to K^* in game $\mathbf{G}_1$ and is equal to K^+ in game $\mathbf{G}_2$. Note that in games $\mathbf{G}_1$ and $\mathbf{G}_2$ the ciphertext ψ^* is never explicitly decrypted, and so A_1 need not submit this for decryption either.

Last, we observe that there is an oracle query machine A_2, whose running time is essentially the same as that of A, such that

$$|\Pr[T_2] - 1/2| = \mathsf{AdvCCA}_{\mathsf{SKE},\mathsf{A}_2}(\lambda).$$

To see this, note that in game $\mathbf{G}_2$ the ciphertext χ^* is produced using the random symmetric encryption key K^+, and also that some other ciphertexts $\chi \neq \chi^*$ are decrypted using K^+, but that the key K^+ plays no other role in game $\mathbf{G}_2$. Thus, in game $\mathbf{G}_2$ the adversary A is essentially just carrying out an adaptive chosen ciphertext attack against SKE. □

Remark 7.2. We stress that it is essential for both KEM and SKE to be secure against adaptive chosen ciphertext attack in order to prove that HPKE is as well. One cannot start with a "weak" KEM and hope to "repair" it with a hybrid construction: doing this may indeed foil some specific attacks, but we know of no way to formally reason about the security of such a scheme. It is also important not to waste the

chosen ciphertext security of KEM by using a "weak" SKE. Indeed, some popular methods of constructing a "digital envelope" use a SKE that may only be secure against passive attacks; even if the resulting composite ciphertext is digitally signed, this does not necessarily provide security against chosen ciphertext attack.

Remark 7.3. In Remarks 6.9 and 6.10, we cautioned that revealing the reason for rejecting a ciphertext may invalidate the proof of security. This is not the case for the above hybrid construction. There are two reasons a ciphertext may be rejected in this construction: either KEM.Decrypt rejects ψ or SKE.Decrypt rejects χ. We leave it to the reader to verify that the above security proof goes through, essentially unchanged, even if the adversary is told the reason for rejecting. This is a nice feature, since in most practical implementations it would be easy to distinguish these two rejection cases, assuming that χ was very long, and that the decryption algorithm halted immediately when KEM.Decrypt rejects.

8. Key derivation functions. In the next section, we will present and analyze a key encapsulation mechanism. The key will be derived by hashing a pair of group elements. In order not to clutter that section, we develop here the notion of such a key derivation function.

Let $\mathcal{G}$ be a computational group scheme specifying a sequence $(\mathbf{S}_\lambda)_{\lambda \in \mathbf{Z}_{\geq 0}}$ of group distributions.

A *key derivation scheme* KDF *associated with* $\mathcal{G}$ specifies two items:

- A family of *key spaces* indexed by $\lambda \in \mathbf{Z}_{\geq 0}$ and $\Gamma \in [\mathbf{S}_\lambda]$. Each such key space is a probability space, denoted $\mathsf{KDF.KeySpace}_{\lambda,\Gamma}$, on bit strings, called *derivation keys*.
  There must exist a probabilistic, polynomial-time algorithm whose output distribution on input 1^λ and Γ is equal to $\mathsf{KDF.KeySpace}_{\lambda,\Gamma}$.
- A family of *key derivation functions* indexed by $\lambda \in \mathbf{Z}_{\geq 0}$, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, and $\mathsf{dk} \in [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}]$, where each such function $\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}$ maps a pair $(a, b) \in G^2$ of group elements to a key K.
  A key K is a bit string of length $\mathsf{KDF.OutLen}(\lambda)$. The parameter $\mathsf{KDF.OutLen}(\lambda)$ should be computable in deterministic polynomial time given 1^λ.
  There must exist a deterministic, polynomial-time algorithm that on input 1^λ, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, $\mathsf{dk} \in [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}]$, and $(a, b) \in G^2$ outputs $\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$.

We now define the security property that we shall require of KDF.

For all 0/1-valued, probabilistic, polynomial-time algorithms A, and for all $\lambda \in \mathbf{Z}_{\geq 0}$, define

$$
\begin{aligned}
&\mathsf{AdvDist}_{\mathsf{KDF},\mathsf{A}}(\lambda) \\
&:= \Big| \Pr[\ \tau = 1 : \Gamma \xleftarrow{R} \mathbf{S}_\lambda;\ \mathsf{dk} \xleftarrow{R} \mathsf{KDF.KeySpace}_{\lambda,\Gamma};\ a, b \xleftarrow{R} G; \\
&\qquad\qquad \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \mathsf{dk}, a, \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b))\ ] \\
&\quad - \Pr[\ \tau = 1 : \Gamma \xleftarrow{R} \mathbf{S}_\lambda;\ \mathsf{dk} \xleftarrow{R} \mathsf{KDF.KeySpace}_{\lambda,\Gamma};\ a \xleftarrow{R} G;\ K \xleftarrow{R} \{0,1\}^{\mathsf{KDF.OutLen}(\lambda)}; \\
&\qquad\qquad \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \mathsf{dk}, a, K)\ ] \Big| .
\end{aligned}
$$

That is, $\mathsf{AdvDist}_{\mathsf{KDF},\mathsf{A}}(\lambda)$ measures the advantage that A has in distinguishing two distributions: in the first it is given $\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$ and in the second it is given a random key K; in both distributions it is given the derivation key dk as well as the

auxiliary group element a, so, in effect, both dk and a are to be regarded as public data.

We shall say that KDF is *secure* if this distinguishing advantage is negligible; i.e.,

> *for all* 0/1*-valued, probabilistic, polynomial-time algorithms* A *the function* $\mathsf{AdvDist}_{\mathsf{KDF},\mathsf{A}}(\lambda)$ *grows negligibly in* λ.

It is also convenient to define a quantity analogous to $\mathsf{AdvDist}_{\mathsf{KDF},\mathsf{A}}(\lambda)$ but conditioned on a fixed group description. For all 0/1-valued, probabilistic, polynomial-time algorithms A, for all $\lambda \in \mathbf{Z}_{\geq 0}$, and all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$,

$$\begin{aligned}
&\mathsf{AdvDist}_{\mathsf{KDF},\mathsf{A}}(\lambda \mid \Gamma) \\
&:= \Big| \Pr[\ \tau = 1 : \mathsf{dk} \xleftarrow{R} \mathsf{KDF.KeySpace}_{\lambda,\Gamma};\ a, b \xleftarrow{R} G; \\
&\qquad\qquad \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \mathsf{dk}, a, \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a,b))\ ] \\
&\quad - \Pr[\ \tau = 1 : \mathsf{dk} \xleftarrow{R} \mathsf{KDF.KeySpace}_{\lambda,\Gamma};\ a \xleftarrow{R} G;\ K \xleftarrow{R} \{0,1\}^{\mathsf{KDF.OutLen}(\lambda)}; \\
&\qquad\qquad \tau \xleftarrow{R} \mathsf{A}(1^\lambda, \Gamma, \mathsf{dk}, a, K)\ ] \Big| .
\end{aligned}$$

8.1. Constructions.

8.1.1. Unconditionally secure constructions. One can build a secure KDF for $\mathcal{G}$ without any assumptions, provided the groups defined by $\mathcal{G}$ are sufficiently large, which they certainly will be in our applications. Indeed, all we need is that KDF is *pairwise independent*.

In our context, we shall say that a KDF is pairwise independent if for all $\lambda \in \mathbf{Z}_{\geq 0}$, for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, for all $a, b, b' \in G$ with $b \neq b'$, the distribution

$$\{(\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a,b), \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a,b')) : \mathsf{dk} \xleftarrow{R} \mathsf{KDF.KeySpace}_{\lambda,\Gamma}\}$$

is the uniform distribution over all pairs of bits strings of length $\mathsf{KDF.OutLen}(\lambda)$.

By the leftover hash lemma [36, 37], it follows that if KDF is pairwise independent, then for all 0/1-valued, probabilistic, polynomial-time algorithms A, for all $\lambda \in \mathbf{Z}_{\geq 0}$, and for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$,

$$\mathsf{AdvDist}_{\mathsf{KDF},\mathsf{A}}(\lambda \mid \Gamma) \leq 2^{-k},$$

where

$$k = \left\lfloor \frac{\lfloor \log_2 q \rfloor - \mathsf{KDF.OutLen}(\lambda)}{2} \right\rfloor .$$

We also point out that fairly efficient pairwise independent functions can be constructed without relying on any intractability assumptions.

8.1.2. Conditionally secure constructions. In practice, to build a key derivation function one might simply use a dedicated cryptographic hash function, such as SHA-1.

In this situation, we will simply be forced to assume that such a KDF is secure. However, such an intractability assumption is not entirely unreasonable. Moreover, a dedicated cryptographic hash function has several potential advantages over a pairwise independent hash function:

- it may not use a key, or it may use a very short key, which may lead to a significant space savings;

- it can usually be evaluated more quickly than a typical pairwise independent hash function;
- it can be safely used to derive output keys that are significantly longer than would be safe to derive with a typical pairwise independent hash function;
- it may, at least heuristically, provide even more security in applications than a typical pairwise independent hash function.

9. The new encryption scheme: Hybrid version.

9.1. Description of the scheme. In this section, we present a hybrid version of our new encryption scheme. Specifically, we present a key encapsulation mechanism $\mathsf{CS3}$, out of which one can easily construct a hybrid encryption scheme, as described in section 7.

The scheme makes use of a computational group scheme $\mathcal{G}$ as described in section 4.1, defining a sequence $(\mathbf{S}_\lambda)_{\lambda \in \mathbf{Z}_{\geq 0}}$ of distributions of group descriptions and providing a sampling algorithm $\hat{S}$, where the output distribution $\hat{S}(1^\lambda)$ closely approximates $\mathbf{S}_\lambda$.

The scheme also makes use of a binary group hashing scheme HF associated with $\mathcal{G}$, as described in section 5.

Finally, the scheme makes use of a key derivation scheme KDF, associated with $\mathcal{G}$, as described in section 8. Note that output key length $\mathsf{CS3.KeyLen}(\lambda)$ of the scheme is equal to $\mathsf{KDF.OutLen}(\lambda)$.

The scheme is described in detail in Figure 9.1.

9.2. Security analysis of the scheme. We shall prove that $\mathsf{CS3}$ is secure against adaptive chosen ciphertext attack if the DDH assumption holds for $\mathcal{G}$ and the TCR assumption holds for HF, and assuming that KDF is a secure key derivation scheme.

As we have done before, for all probabilistic, polynomial-time oracle query machines A and for all $\lambda \in \mathbf{Z}_{\geq 0}$, we let $Q_{\mathsf{A}}(\lambda)$ be an upper bound on the number of decryption oracle queries made by A on input 1^λ. We assume that $Q_{\mathsf{A}}(\lambda)$ is a strict bound in the sense that it holds regardless of the probabilistic choices of A and regardless of the responses to its oracle queries from its environment.

THEOREM 9.1. *If the DDH assumption holds for $\mathcal{G}$ and the TCR assumption holds for HF, and assuming that KDF is a secure key derivation scheme, then $\mathsf{CS3}$ is secure against adaptive chosen ciphertext attack*

In particular, for all probabilistic, polynomial-time oracle query machines A, there exist probabilistic algorithms A_1, A_2, and A_3, whose running times are essentially the same as that of A, such that the following holds. For all $\lambda \in \mathbf{Z}_{\geq 0}$ and all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we have

$$\begin{aligned} \mathsf{AdvCCA}_{\mathsf{CS3},\mathsf{A}}(\lambda \mid \Gamma) \;\leq\; & \mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}_1}(\lambda \mid \Gamma)\mathsf{AdvTCR}_{\mathsf{HF},\mathsf{A}_2}(\lambda \mid \Gamma) \\ & + \mathsf{AdvDist}_{\mathsf{KDF},\mathsf{A}_3}(\lambda \mid \Gamma) + (Q_{\mathsf{A}}(\lambda) + 3)/q. \end{aligned} \tag{9.1}$$

To prove Theorem 9.1, let us fix a probabilistic, polynomial-time oracle query machine A, the value of the security parameter $\lambda \in \mathbf{Z}_{\geq 0}$, and the group description $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$.

The proof follows the same line of argument as the proof of Theorem 6.1, and we will at several places appeal to arguments in that proof so as to avoid unnecessary repetition.

The attack game is as described in section 7.1.2. We now discuss the relevant random variables in this game.

Key Generation: On input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, compute

$$\Gamma[\hat{G}, G, g, q] \xleftarrow{R} \hat{S}(1^\lambda);\ \mathsf{hk} \xleftarrow{R} \mathsf{HF.KeySpace}_{\lambda,\Gamma};\ \mathsf{dk} \xleftarrow{R} \mathsf{KDF.KeySpace}_{\lambda,\Gamma};$$
$$w \xleftarrow{R} \mathbf{Z}_q^*;\ x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{R} \mathbf{Z}_q;$$
$$\hat{g} \leftarrow g^w;\ e \leftarrow g^{x_1}\hat{g}^{x_2};\ f \leftarrow g^{y_1}\hat{g}^{y_2};\ h \leftarrow g^{z_1}\hat{g}^{z_2}$$

and output the public key $\mathsf{PK} = (\Gamma, \mathsf{hk}, \mathsf{dk}, \hat{g}, e, f, h)$ and the secret key $\mathsf{SK} = (\Gamma, \mathsf{hk}, \mathsf{dk}, x_1, x_2, y_1, y_2, z_1, z_2)$.

Encryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a public key

$$\mathsf{PK} = (\Gamma[\hat{G}, G, g, q], \mathsf{hk}, \mathsf{dk}, \hat{g}, e, f, h) \in [\mathbf{S}_\lambda] \times [\mathsf{HF.KeySpace}_{\lambda,\Gamma}] \times [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}] \times G^4,$$

compute

E1: $u \xleftarrow{R} \mathbf{Z}_q$;
E2: $a \leftarrow g^u$;
E3: $\hat{a} \leftarrow \hat{g}^u$;
E4: $b \leftarrow h^u$;
E5: $K \leftarrow \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$;
E6: $v \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a})$;
E7: $d \leftarrow e^u f^{uv}$

and output the symmetric key K and the ciphertext $\psi = (a, \hat{a}, d)$.

Decryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a secret key

$$\mathsf{SK} = (\Gamma[\hat{G}, G, g, q], \mathsf{hk}, \mathsf{dk}, x_1, x_2, y_1, y_2, z_1, z_2) \in [\mathbf{S}_\lambda] \times [\mathsf{HF.KeySpace}_{\lambda,\Gamma}] \times [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}] \times \mathbf{Z}_q^6,$$

along with a ciphertext ψ, do the following.

D1: Parse ψ as a 3-tuple $(a, \hat{a}, d) \in \hat{G}^3$; output reject and halt if ψ is not of this form.
D2: Test if a and $\hat{a}$ belong to G; output reject and halt if this is not the case.
D3: Compute $v \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a})$.
D4: Test if $d = a^{x_1+y_1 v}\hat{a}^{x_2+y_2 v}$; output reject and halt if this is not the case.
D5: Compute $b \leftarrow a^{z_1}\hat{a}^{z_2}$.
D6: Compute $K \leftarrow \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$ and output the symmetric key K.

FIG. 9.1. *The key encapsulation mechanism* CS3.

Suppose that the public key is $(\Gamma, \mathsf{hk}, \mathsf{dk}, \hat{g}, e, f, h)$ and that the secret key is $(\Gamma, \mathsf{hk}, \mathsf{dk}, x_1, x_2, y_1, y_2, z_1, z_2)$. Let $w := \log_g \hat{g}$, and define $x, y, z \in \mathbf{Z}_q$ as follows:

$$x := x_1 + x_2 w,\ y := y_1 + y_2 w,\ z := z_1 + z_2 w.$$

As a notational convention, whenever a particular ciphertext ψ is under consideration in some context, the following values are also implicitly defined in that context:

- $a, \hat{a}, d \in G$, where $\psi = (a, \hat{a}, d)$;
- $u, \hat{u}, v, s \in \mathbf{Z}_q$, where

$$u := \log_g a,\ \hat{u} := \log_{\hat{g}} \hat{a},\ v := \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a}),\quad s := \log_g d.$$

For the target ciphertext ψ^*, we also denote by $a^*, \hat{a}^*, d^* \in G$, and $u^*, \hat{u}^*, v^*, s^* \in \mathbf{Z}_q$ the corresponding values.

The probability space defining the attack game is then determined by the following, mutually independent, random variables:

- the coin tosses of A;
- the values $\mathsf{hk}, \mathsf{dk}, w, x_1, x_2, y_1, y_2, z_1, z_2$ generated by the key generation algorithm;
- the values $\tau \in \{0,1\}$, $K^+ \in \{0,1\}^{\mathsf{KDF.OutLen}(\lambda)}$, and $u^* \in \mathbf{Z}_q$ generated by the encryption oracle in the attack game.

Let $\mathbf{G}_0$ be the original attack game, let $\hat{\tau} \in \{0,1\}$ denote the output of A, and let T_0 be the event that $\tau = \hat{\tau}$ in $\mathbf{G}_0$ so that $\mathsf{AdvCCA}_{\mathsf{CS3},\mathsf{A}}(\lambda \mid \Gamma) = |\Pr[T_0] - 1/2|$.

As in the proof of Theorem 6.1, we shall define a sequence of modified games $\mathbf{G}_i$, for $i = 1, 2, \ldots$, where in game $\mathbf{G}_i$ the event T_i will be the event corresponding to event T_0 but in game $\mathbf{G}_i$. The overall structure of the proof will differ a bit from that of Theorem 6.1, even though many of the low-level details will be very similar. Indeed, the proof of this theorem is conceptually a bit simpler (even though there are more steps) than that of Theorem 6.1, since the inputs to $\mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}$ in the encryption oracle are independent of any quantities computed by the adversary; we also save a term of $1/q$ in (9.1) because of this (compare with (6.1) in Theorem 6.1).

Game $\mathbf{G_1}$. We now modify game $\mathbf{G}_0$ to obtain a new game $\mathbf{G}_1$. These two games are identical, except that instead of using the encryption algorithm as given to compute the target ciphertext ψ^* we use a modified encryption algorithm in which steps **E4** and **E7** are replaced by

E4′: $b \leftarrow a^{z_1}\hat{a}^{z_2}$;

E7′: $d \leftarrow a^{x_1+y_1 v}\hat{a}^{x_2+y_2 v}$.

By the same reasoning as in the proof of Theorem 6.1, we have

$$\Pr[T_1] = \Pr[T_0].$$

Game $\mathbf{G_2}$. We again modify the encryption oracle, replacing step **E3** by

E3′: $\hat{u} \xleftarrow{R} \mathbf{Z}_q$; $\hat{a} \leftarrow \hat{g}^{\hat{u}}$.

By the same reasoning as in the proof of Theorem 6.1, one sees that there exists a probabilistic algorithm A_1, whose running time is essentially the same as that of A, such that

$$|\Pr[T_2] - \Pr[T_1]| \leq \mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}_1}(\lambda \mid \Gamma) + 2/q.$$

Note that unlike game $\mathbf{G}_2$ in the proof of Theorem 6.1, we do not impose the restriction $u^* \neq \hat{u}^*$ just yet; it is technically convenient to defer this until later. This is why the term $2/q$ appears in the above bound rather than $3/q$.

Game $\mathbf{G_3}$. This game is the same as game $\mathbf{G}_2$, except for the following modification.

We modify the decryption oracle so that it applies the following *special rejection rule*: if the adversary submits a ciphertext ψ for decryption at a point in time after the encryption oracle has been invoked such that $(a, \hat{a}) \neq (a^*, \hat{a}^*)$ but $v = v^*$, then the decryption oracle immediately outputs reject and halts (before executing step **D4′**).

We claim that there exists a probabilistic algorithm A_2, whose running time is essentially the same as that of A, such that

$$|\Pr[T_3] - \Pr[T_2]| \leq \mathsf{AdvTCR}_{\mathsf{HF},\mathsf{A}_2}(\lambda \mid \Gamma).$$

This follows from reasoning very similar to the proof of Lemma 6.5 in the analysis of game $\mathbf{G}_5$ in the proof of Theorem 6.1. Observe that we can impose the special

rejection rule already in this game, rather than deferring to a later game as in the proof of Theorem 6.1, because, as we mentioned above, the inputs to $\mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}$ in the encryption oracle are independent of any quantities computed by the adversary.

Game $\mathbf{G_4}$. We again modify the encryption oracle, replacing step **E3′** by

E3″: $\hat{u} \xleftarrow{R} \mathbf{Z}_q \setminus \{u\}$; $\hat{a} \leftarrow \hat{g}^{\hat{u}}$.

It is easy to verify that

$$|\Pr[T_4] - \Pr[T_3]| \leq 1/q.$$

Game $\mathbf{G_5}$. In this game, we modify the decryption oracle in game $\mathbf{G}_4$, replacing steps **D4** and **D5** by

D4′: test if $\hat{a} = a^w$ and $d = a^{x+yv}$; output reject and halt if this is not the case;

D5′: $b \leftarrow a^z$.

Let R_5 be the event that in game $\mathbf{G}_5$ some ciphertext ψ is submitted to the decryption oracle that is rejected in step **D4′** but that would have passed the test in step **D4**.

It is clear that

$$|\Pr[T_5] - \Pr[T_4]| \leq \Pr[R_5].$$

We also claim that

$$\Pr[R_5] \leq Q_{\mathsf{A}}(\lambda)/q.$$

This follows from reasoning analogous to that in Lemma 6.6 (in game $\mathbf{G}_5$ in the proof of Theorem 6.1).

Game $\mathbf{G_6}$. We again modify the algorithm used by the encryption oracle, replacing step **E4′** by

E4″: $r \xleftarrow{R} \mathbf{Z}_q$; $b \leftarrow g^r$.

By reasoning analogous to that in the analysis of game $\mathbf{G}_4$ in the proof of Theorem 6.1, one can easily show that

$$\Pr[T_6] = \Pr[T_5].$$

Game $\mathbf{G_7}$. In this game, we modify the encryption oracle, replacing step **E5** of the encryption algorithm by

E5′: $K \xleftarrow{R} \{0,1\}^{\mathsf{KDF.OutLen}(\lambda)}$.

It is straightforward to see that there exists a probabilistic algorithm A_3, whose running time is essentially the same as that of A, such that

$$|\Pr[T_7] - \Pr[T_6]| \leq \mathsf{AdvDist}_{\mathsf{KDF},\mathsf{A}_3}(\lambda \mid \Gamma).$$

Furthermore, it is clear by construction that

$$\Pr[T_7] = 1/2.$$

That completes the proof of Theorem 9.1.

9.3. Two variations. One can easily modify scheme CS3 to obtain two variants, which we call CS3a and CS3b, that are analogous to the variations CS1a and CS1b of CS1, discussed in section 6.3. Only the key generation and decryption algorithms differ. The details are presented in Figures 9.2 and 9.3.

Key Generation: On input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, compute

$$\Gamma[\hat{G}, G, g, q] \xleftarrow{R} \hat{S}(1^\lambda);\ \mathsf{hk} \xleftarrow{R} \mathsf{HF.KeySpace}_{\lambda,\Gamma};\ \mathsf{dk} \xleftarrow{R} \mathsf{KDF.KeySpace}_{\lambda,\Gamma};$$
$$w \xleftarrow{R} \mathbf{Z}_q^*;\ x_1, x_2, y_1, y_2, z \xleftarrow{R} \mathbf{Z}_q;$$
$$\hat{g} \leftarrow g^w;\ e \leftarrow g^{x_1}\hat{g}^{x_2};\ f \leftarrow g^{y_1}\hat{g}^{y_2};\ h \leftarrow g^z$$

and output the public key $\mathsf{PK} = (\Gamma, \mathsf{hk}, \mathsf{dk}, \hat{g}, e, f, h)$ and the secret key $\mathsf{SK} = (\Gamma, \mathsf{hk}, \mathsf{dk}, x_1, x_2, y_1, y_2, z)$.

Decryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a secret key

$$\mathsf{SK} = (\Gamma[\hat{G}, G, g, q], \mathsf{hk}, \mathsf{dk}, x_1, x_2, y_1, y_2, z)$$
$$\in [\mathbf{S}_\lambda] \times [\mathsf{HF.KeySpace}_{\lambda,\Gamma}] \times [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}] \times \mathbf{Z}_q^5,$$

along with a ciphertext ψ, do the following.

D1: Parse ψ as a 3-tuple $(a, \hat{a}, d) \in \hat{G}^3$; output reject and halt if ψ is not of this form.
D2: Test if a and $\hat{a}$ belong to G; output reject and halt if this is not the case.
D3: Compute $v \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a})$.
D4: Test if $d = a^{x_1+y_1 v}\hat{a}^{x_2+y_2 v}$; output reject and halt if this is not the case.
D5′: Compute $b \leftarrow a^z$.
D6: Compute $K \leftarrow \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$ and output the symmetric key K.

FIG. 9.2. *Key generation and decryption algorithms for* CS3a.

Key Generation: On input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, compute

$$\Gamma[\hat{G}, G, g, q] \xleftarrow{R} \hat{S}(1^\lambda);\ \mathsf{hk} \xleftarrow{R} \mathsf{HF.KeySpace}_{\lambda,\Gamma};\ \mathsf{dk} \xleftarrow{R} \mathsf{KDF.KeySpace}_{\lambda,\Gamma};$$
$$w \xleftarrow{R} \mathbf{Z}_q^*;\ x, y, z \xleftarrow{R} \mathbf{Z}_q;$$
$$\hat{g} \leftarrow g^w;\ e \leftarrow g^x;\ f \leftarrow g^y;\ h \leftarrow g^z$$

and output the public key $\mathsf{PK} = (\Gamma, \mathsf{hk}, \mathsf{dk}, \hat{g}, e, f, h)$ and the secret key $\mathsf{SK} = (\Gamma, \mathsf{hk}, \mathsf{dk}, x, y, z)$.

Decryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a secret key

$$\mathsf{SK} = (\Gamma[\hat{G}, G, g, q], \mathsf{hk}, \mathsf{dk}, x, y, z) \in [\mathbf{S}_\lambda] \times [\mathsf{HF.KeySpace}_{\lambda,\Gamma}] \times [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}] \times \mathbf{Z}_q^3,$$

along with a ciphertext ψ, do the following.

D1: Parse ψ as a 3-tuple $(a, \hat{a}, d) \in \hat{G}^3$; output reject and halt if ψ is not of this form.
D2′: Test if a belongs to G; output reject and halt if this is not the case.
D3: Compute $v \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a})$.
D4′: Test if $\hat{a} = a^w$ and $d = a^{x+yv}$; output reject and halt if this is not the case.
D5′: Compute $b \leftarrow a^z$.
D6: Compute $K \leftarrow \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$ and output the symmetric key K.

FIG. 9.3. *Key generation and decryption algorithms for* CS3b.

Remark 9.1. Scheme CS3b is essentially the same scheme that was originally presented in [56]. This scheme is the most efficient scheme among all those presented in this paper. It is also attractive in that it yields a public-key encryption scheme

with an unrestricted message space. Moreover, this scheme has some other attractive security properties that will be examined in section 10.

Remark 9.2. Analogous to Remark 6.7, we do not have to separately test if $\hat{a}$ belongs to the subgroup G in step **D2′** of the decryption algorithm of CS3b, and we may test if a belongs to G in some cases by testing if $a^q = 1_G$.

Remark 9.3. Analogous to Remark 6.8, in scheme CS3b the decryption algorithm has to compute either three or four (if we test if $a^q = 1_G$) powers of a, and special algorithmic techniques can be exploited to do this.

Remark 9.4. Analogous to Remarks 6.9 and 6.10, it is strongly recommended to *always* compute both exponentiations in step **D4′** of CS3b before rejecting the ciphertext and to not reveal the precise reason why any ciphertext was rejected. Again, we know of no actual attack given this information, and, in fact, it seems very unlikely—see section 10.7 below.

The following theorem can proved using an argument almost identical to the argument that was used to prove Theorem 6.10. We leave it to the reader to verify this.

THEOREM 9.2. *If the DDH assumption holds for $\mathcal{G}$ and the TCR assumption holds for* HF*, and assuming that* KDF *is a secure key derivation scheme, then* CS3a *and* CS3b *are secure against adaptive chosen ciphertext attack.*

In particular, for all probabilistic, polynomial-time oracle query machines A*, for all $\lambda \in \mathbf{Z}_{\geq 0}$, and for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we have*

$$|\mathsf{AdvCCA}_{\mathsf{CS3a},\mathsf{A}}(\lambda \mid \Gamma) - \mathsf{AdvCCA}_{\mathsf{CS3},\mathsf{A}}(\lambda \mid \Gamma)| \leq Q_{\mathsf{A}}(\lambda)/q$$

and

$$|\mathsf{AdvCCA}_{\mathsf{CS3b},\mathsf{A}}(\lambda \mid \Gamma) - \mathsf{AdvCCA}_{\mathsf{CS3},\mathsf{A}}(\lambda \mid \Gamma)| \leq Q_{\mathsf{A}}(\lambda)/q.$$

10. Further security considerations of scheme CS3b. The key encapsulation mechanism CS3b, which was described and analyzed in section 9.3, has some other interesting security properties, which we discuss in this section.

The main results we present here are the following. First, we show that CS3b is no less secure than a more traditional key encapsulation mechanism that is a hashed variant of ElGamal encryption, which we call HEG. Second, we also show that CS3b is secure in the random oracle model (viewing KDF as a random oracle) if the CDH and TCR assumptions hold. Along the way, we also give a security analysis of HEG in the random oracle model, based on a rather nonstandard intractability assumption.

10.1. Hashed ElGamal key encapsulation. We begin by presenting a fairly traditional version of ElGamal key encapsulation, which we call HEG.

The scheme makes use of a computational group scheme $\mathcal{G}$ as described in section 4.1, defining a sequence $(\mathbf{S}_\lambda)_{\lambda \in \mathbf{Z}_{\geq 0}}$ of distributions of group descriptions and providing a sampling algorithm $\hat{S}$, where the output distribution $\hat{S}(1^\lambda)$ closely approximates $\mathbf{S}_\lambda$.

Also, the scheme makes use of a key derivation scheme KDF, associated with $\mathcal{G}$, as described in section 8. Note that output key length $\mathsf{EG.KeyLen}(\lambda)$ of the scheme is equal to $\mathsf{KDF.OutLen}(\lambda)$.

The scheme is described in detail in Figure 10.1.

10.2. The random oracle model. We will analyze the security of both schemes HEG and CS3b in the random oracle model. In this approach, a cryptographic hash function—in this case KDF—is modeled for the purposes of analysis as a "black box"

Key Generation: On input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, compute

$$\Gamma[\hat{G}, G, g, q] \xleftarrow{R} \hat{S}(1^\lambda);\ \mathsf{dk} \xleftarrow{R} \mathsf{KDF.KeySpace}_{\lambda,\Gamma};\ z \xleftarrow{R} \mathbf{Z}_q;\ h \leftarrow g^z$$

and output the public key $\mathsf{PK} = (\Gamma, \mathsf{dk}, h)$ and the secret key $\mathsf{SK} = (\Gamma, \mathsf{dk}, z)$.

Encryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a public key

$$\mathsf{PK} = (\Gamma[\hat{G}, G, g, q], \mathsf{dk}, h) \in [\mathbf{S}_\lambda] \times [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}] \times G,$$

compute

E1: $u \xleftarrow{R} \mathbf{Z}_q$;
E2: $a \leftarrow g^u$;
E3: $b \leftarrow h^u$;
E4: $K \leftarrow \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$

and output the symmetric key K and the ciphertext $\psi = a$.

Decryption: Given 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, a secret key

$$\mathsf{SK} = (\Gamma[\hat{G}, G, g, q], \mathsf{dk}, z) \in [\mathbf{S}_\lambda] \times [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}] \times \mathbf{Z}_q,$$

along with a ciphertext ψ, do the following.

D1: Parse ψ as a group element $a \in \hat{G}$; output reject and halt if ψ is not of this form.
D2: Test if a belongs to G; output reject and halt if this is not the case.
D3: Compute $b \leftarrow a^z$.
D4: Compute $K \leftarrow \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$ and output the symmetric key K.

FIG. 10.1. *The key encapsulation mechanism* HEG.

containing a random function to which the adversary and the algorithms implementing the cryptosystem have "oracle access." This approach has been used implicitly and informally for some time; however, it was formalized by Bellare and Rogaway [9] and has subsequently been used quite a bit in the cryptographic research community.

More precisely, we shall analyze the security of the scheme HEG and later CS3b in an idealized model of computation where for all $\lambda \in \mathbf{Z}_{\geq 0}$, all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, all $\mathsf{dk} \in [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}]$, and all $a, b \in G$, we treat the values $\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$ as mutually independent, random bit strings of length $\mathsf{KDF.OutLen}(\lambda)$; moreover, the only way to obtain the value of $\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$ is to explicitly query an oracle with input $(\lambda, \Gamma, \mathsf{dk}, a, b)$. Actually, to be complete, we allow Γ, dk, a, and b to range over arbitrary bit strings, regardless of whether these are valid encodings of appropriate objects. Since in any of our applications only a finite number of the values $\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$ will be relevant, experiments based on these values can be modeled using finite probability spaces.

When considering an adversary A that is carrying out an adaptive chosen ciphertext attack in the random oracle model, in addition to the usual types of oracle queries that A makes, the adversary A is also allowed to query the random oracle representing KDF. We shall denote by $Q'_{\mathsf{A}}(\lambda)$ a strict upper bound on the number of random oracle queries that A makes for a given value of the security parameter λ; as usual, this bound should hold regardless of the environment in which A actually runs.

10.3. CS3b is at least as secure as HEG. We now show that the scheme CS3b is at least as secure as HEG.

THEOREM 10.1. *If scheme* HEG *is secure against adaptive chosen ciphertext attack, then so is* CS3b*; moreover, this implication holds in either the standard or random oracle models.*

In particular, for all probabilistic, polynomial-time oracle query machines A*, there exists another oracle query machine* A_1*, whose running time is essentially the same as that of* A*, such that for all* $\lambda \in \mathbf{Z}_{\geq 0}$ *and all* $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$ *we have*

$$\mathsf{AdvCCA}_{\mathsf{CS3b},\mathsf{A}}(\lambda \mid \Gamma) \leq \mathsf{AdvCCA}_{\mathsf{HEG},\mathsf{A}_1}(\lambda \mid \Gamma);$$

moreover, $Q_{\mathsf{A}_1}(\lambda) \leq Q_{\mathsf{A}}(\lambda)$ *and (in the random oracle model)* $Q'_{\mathsf{A}_1}(\lambda) \leq Q'_{\mathsf{A}}(\lambda)$.

Proof. Fix A, λ, and $\Gamma[\hat{G}, G, g, q]$ as above. We construct an adversary A_1 that attacks HEG. The adversary A_1 makes use of A by providing an environment for A as follows.

First, suppose that A_1 is given a public key (Γ, dk, h) for scheme HEG, where Γ is fixed as above. Adversary A_1 then "dresses up" the HEG public key to look like a CS3b public key; namely, A_1 computes

$$\mathsf{hk} \xleftarrow{R} \mathsf{HF.KeySpace}_{\lambda,\Gamma};\ w \xleftarrow{R} \mathbf{Z}_q^*;\ x, y \xleftarrow{R} \mathbf{Z}_q;\ \hat{g} \leftarrow g^w;\ e \leftarrow g^x;\ f \leftarrow g^y$$

and presents A with the CS3b public key

$$(\Gamma, \mathsf{hk}, \mathsf{dk}, \hat{g}, e, f, h).$$

Second, whenever A submits a CS3b ciphertext $(a, \hat{a}, d) \in \hat{G}^3$ to the decryption oracle, adversary A_1 first performs the validity tests of the decryption algorithm of CS3b, making use of the values hk, w, x, y generated above; if these tests pass, then A_1 invokes the decryption oracle of HEG with input a.

Third, when A invokes the encryption oracle of CS3b, adversary A_1 does the following. It invokes the encryption oracle of HEG, obtaining a ciphertext $a^* \in G$ and a key $K^\dagger$. It then "dresses up" a^* to look like a CS3b ciphertext; namely, it computes

$$\hat{a}^* \leftarrow (a^*)^w;\ v^* \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a^*, \hat{a}^*);\ d^* \leftarrow (a^*)^{x+yv^*}$$

and presents A with the CS3b ciphertext $(a^*, \hat{a}^*, d^*)$ along with the key $K^\dagger$.

Fourth, when A terminates and outputs a value, A_1 also terminates and outputs the same value.

That completes the description of the adversary A_1.

One has to check that A_1 carries out a legal adaptive chosen ciphertext attack in the sense that it should not attempt to submit the target ciphertext itself to the decryption oracle, subsequent to the invocation of the encryption oracle. Consider a ciphertext a submitted by A_1 to the decryption oracle. This was derived from a valid CS3b ciphertext $(a, \hat{a}, d)$ submitted by A to the decryption oracle. By the construction, it is easy to see that if $a = a^*$, then, in fact, $(a, \hat{a}, d) = (a^*, \hat{a}^*, d^*)$, which cannot happen if A itself carries out a legal attack.

Since the simulation by A_1 above is perfect, it is clear that whatever advantage A has in guessing the hidden bit, adversary A_1 has precisely the same advantage. It is also clear by construction that $Q_{\mathsf{A}_1}(\lambda) \leq Q_{\mathsf{A}}(\lambda)$ and in the random oracle model that $Q'_{\mathsf{A}_1}(\lambda) \leq Q'_{\mathsf{A}}(\lambda)$. □

10.4. The security of HEG in the random oracle model. As for the security of HEG, even in the random oracle model we do not know how to prove a very strong result. We content ourselves with a proof that the scheme HEG is secure against

adaptive chosen ciphertext attack in the random oracle model, provided the CDH assumption holds *relative to an oracle for the DDH problem.*

More precisely, for all probabilistic, polynomial-time oracle query machines A and for all $\lambda \in \mathbf{Z}_{\geq 0}$, we define

$$\mathsf{AdvCDH}^*_{\mathcal{G},\mathsf{A}}(\lambda) := \Pr[\ c = g^{xy} : \Gamma[\hat{G}, G, g, q] \xleftarrow{R} \mathbf{S}_\lambda;\ x \xleftarrow{R} \mathbf{Z}_q;\ y \xleftarrow{R} \mathbf{Z}_q; \\ c \xleftarrow{R} \mathsf{A}^{\mathsf{DHP}_{\lambda,\Gamma}}(1^\lambda, \Gamma, g^x, g^y)\ ],$$

where the notation $\mathsf{A}^{\mathsf{DHP}_{\lambda,\Gamma}}(\cdots)$ signifies that A runs with access to an oracle for the Diffie–Hellman predicate $\mathsf{DHP}_{\lambda,\Gamma}$ defined in section 4.3.3.

We say that the CDH assumption for $\mathcal{G}$ holds relative to an oracle for the DDH problem if

> *for all probabilistic, polynomial-time oracle query machines* A *the function* $\mathsf{AdvCDH}^*_{\mathcal{G},\mathsf{A}}(\lambda)$ *is negligible in* λ.

For all probabilistic, polynomial-time oracle query machines A, for all $\lambda \in \mathbf{Z}_{\geq 0}$, and for all $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, we also define

$$\mathsf{AdvCDH}^*_{\mathcal{G},\mathsf{A}}(\lambda \mid \Gamma) := \Pr[\ c = g^{xy} : x \xleftarrow{R} \mathbf{Z}_q;\ y \xleftarrow{R} \mathbf{Z}_q;\ c \xleftarrow{R} \mathsf{A}^{\mathsf{DHP}_{\lambda,\Gamma}}(1^\lambda, \Gamma, g^x, g^y)\ ].$$

On the one hand, it has been proven (see [42]) that in a "generic" model of computation the CDH problem remains hard even in the presence of a DDH oracle, thus giving some justification for this assumption; on the other hand, if one works with the special elliptic curves discussed in the paper [38] already mentioned in section 4.4, then the DDH oracle actually has an efficient implementation, even though the CDH problem still appears hard.

THEOREM 10.2. *The scheme* HEG *is secure in the random oracle model if the CDH assumption for* $\mathcal{G}$ *holds relative to an oracle for the DDH problem.*

In particular, for all probabilistic, polynomial-time oracle query machines A*, there exists an oracle query machine* A_1*, whose running time is essentially the same as that of* A*, such that for all* $\lambda \in \mathbf{Z}_{\geq 0}$ *and for all* $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$ *we have*

$$\mathsf{AdvCCA}_{\mathsf{HEG},\mathsf{A}}(\lambda \mid \Gamma) \leq \mathsf{AdvCDH}^*_{\mathcal{G},\mathsf{A}_1}(\lambda \mid \Gamma) + Q_{\mathsf{A}}(\lambda)/q;$$

moreover, the number of DDH-oracle queries made by A_1 *is bounded by* $Q'_{\mathsf{A}}(\lambda)$.

To prove Theorem 10.2, let us fix A, λ, and $\Gamma[\hat{G}, G, g, q]$. The attack game is as described in section 7.1.2.

We begin by describing the relevant random variables in the attack game. The public key is (Γ, dk, h) and the secret key is (Γ, dk, z).

For a given ciphertext ψ, we let $a \in G$ denote the corresponding group element, and we let $b := a^z$, $u := \log_g a$, and $K := \mathsf{KDF}^{\lambda,\Gamma}_{\mathsf{dk}}(a, b)$. Note also that $b = a^u$. For the target ciphertext ψ^*, we let a^*, b^*, u^*, and K^* denote the corresponding values.

The encryption oracle also generates values $\tau \in \{0,1\}$ and $K^+ \in \{0,1\}^{\mathsf{KDF.OutLen}(\lambda)}$.

Let $\mathbf{G}_0$ be the original attack game, let $\hat{\tau}$ denote the output of A, and let T_0 be the event that $\tau = \hat{\tau}$ so that $\mathsf{AdvCCA}_{\mathsf{HEG},\mathsf{A}}(\lambda \mid \Gamma) = |\Pr[T_0] - 1/2|$.

As usual, we define a sequence of games $\mathbf{G}_1$, $\mathbf{G}_2$, etc., and in game $\mathbf{G}_i$ for $i \geq 1$ we define T_i to be the event in game $\mathbf{G}_i$ corresponding to event T_0 in game $\mathbf{G}_0$.

Game $\mathbf{G}_1$. We modify game $\mathbf{G}_0$ as follows. First, we run the encryption oracle at the beginning of the attack game, but we give the results of this to the adversary only when it actually invokes the encryption oracle. This is a purely conceptual change, since the adversary provides no input to the encryption oracle. Second, if the adversary ever submits a ciphertext $\psi = \psi^*$ to the decryption oracle *before* the

encryption algorithm is invoked, we abort the game immediately before responding to this decryption oracle invocation (the environment, say, goes silent at this point).

Let F_1 be the event that game $\mathbf{G}_1$ is aborted as above. It is clear that $\Pr[F_1] \leq Q_{\mathsf{A}}(\lambda)/q$. It is also clear that games $\mathbf{G}_0$ and $\mathbf{G}_1$ proceed identically until event F_1 occurs, and so, by Lemma 6.2, we have $|\Pr[T_1] - \Pr[T_0]| \leq \Pr[F_1]$.

Game $\mathbf{G}_2$. We next modify game $\mathbf{G}_1$ as follows. If the adversary ever queries the random oracle to obtain the value of $\mathsf{KDF}^{\lambda,\Gamma}_{\mathsf{dk}}(a^*, b^*)$, we immediately abort the game before responding to this random oracle invocation.

It is easy to see that $\Pr[T_2] = 1/2$. This follows directly from the fact that in game $\mathbf{G}_2$ the value of $\mathsf{KDF}^{\lambda,\Gamma}_{\mathsf{dk}}(a^*, b^*)$ is obtained from the random oracle only by the encryption oracle: the adversary never queries the random oracle directly at this point, nor does the decryption oracle.

Let F_2 be the event that game $\mathbf{G}_2$ is aborted as above. It is clear that $|\Pr[T_2] - \Pr[T_1]| \leq \Pr[F_2]$, so it suffices to bound $\Pr[F_2]$.

We claim that $\Pr[F_2] = \mathsf{AdvCDH}^*_{\mathcal{G},\mathsf{A}_1}(\lambda \mid \Gamma)$ for an oracle query machine A_1 whose running time and number of oracle queries are bounded as in the statement of the theorem.

We now describe A_1. It takes as input 1^λ, $\Gamma[\hat{G}, G, g, q]$, along with group elements $a^*, h \in G$, and attempts to compute $b^* \in G$ such that $\mathsf{DHP}_{\lambda,\Gamma}(h, a^*, b^*) = 1$. The machine A_1 has access to an oracle for the function $\mathsf{DHP}_{\lambda,\Gamma}$.

Machine A_1 simulates the environment of game $\mathbf{G}_2$ for A as follows. It first computes $\mathsf{dk} \xleftarrow{R} \mathsf{KDF.KeySpace}_{\lambda,\Gamma}$ and gives A the public key (Γ, dk, h). For the target ciphertext, it of course sets $\psi^* := a^*$. For the other output $K^\dagger$ of the encryption oracle, A_1 simply generates this as a random bit string of length $\mathsf{KDF.OutLen}(\lambda)$.

Machine A_1 also needs to simulate the responses to the random oracle and decryption oracle queries. For the random oracle queries, the only values that are relevant are those corresponding to the given values of λ, Γ, and dk.

For the irrelevant random oracle queries, A_1 simply maintains a set of input/output pairs, generating outputs at random as necessary.

Machine A_1 processes relevant random oracle queries using the following data structures:

- a set $\mathcal{V}_1$ of triples (a, b, K), with $a, b \in G$ and $K \in \{0,1\}^{\mathsf{KDF.OutLen}(\lambda)}$, initially empty; this will contain those triples (a, b, K) for which A_1 has assigned the value K to $\mathsf{KDF}^{\lambda,\Gamma}_{\mathsf{dk}}(a, b)$;
- a set $\mathcal{V}_2$ of pairs (a, b), with $a, b \in G$, initially empty; this will contain precisely those pairs (a, b) such that $(a, b, K) \in \mathcal{V}_1$ for some K, and $\mathsf{DHP}_{\lambda,\Gamma}(h, a, b) = 1$;
- a set $\mathcal{V}_3$ of pairs (a, K), with $a \in G$ and $K \in \{0,1\}^{\mathsf{KDF.OutLen}(\lambda)}$, initially empty; this will contain pairs (a, K) for which A_1 has assigned the value K to $\mathsf{KDF}^{\lambda,\Gamma}_{\mathsf{dk}}(a, b)$ for $b \in G$ with $\mathsf{DHP}_{\lambda,\Gamma}(h, a, b) = 1$, even though A_1 does not actually know the value of b.

Given a request for the value $\mathsf{KDF}^{\lambda,\Gamma}_{\mathsf{dk}}(a, b)$, machine A_1 does the following:

- It tests if $(a, b, K) \in \mathcal{V}_1$ for some K. If so (which means that A has queried the value $\mathsf{KDF}^{\lambda,\Gamma}_{\mathsf{dk}}(a, b)$ before), it returns K as the value of $\mathsf{KDF}^{\lambda,\Gamma}_{\mathsf{dk}}(a, b)$; otherwise, it continues.
- It invokes its own DDH-oracle to determine if $\mathsf{DHP}_{\lambda,\Gamma}(h, a, b) = 1$.
- If $\mathsf{DHP}_{\lambda,\Gamma}(h, a, b) = 1$, then we have the following:
  - If $a = a^*$, it halts and outputs the solution $b^* := b$ to the given problem instance (this corresponds to the early-abort rule introduced in game $\mathbf{G}_2$); otherwise, it continues.

– It adds the pair (a, b) to the set $\mathcal{V}_2$.
– If $(a, K) \in \mathcal{V}_3$ for some K, then it adds the triple (a, b, K) to $\mathcal{V}_1$ and returns K as the value of $\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$; otherwise, it continues.

- It generates K as a random bit string of length $\mathsf{KDF.OutLen}(\lambda)$, adds the triple (a, b, K) to $\mathcal{V}_1$, and returns K as the value of $\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$.

Machine A_1 processes decryption oracle queries as follows. Suppose it is given a ciphertext ψ, with $a \in G$ the corresponding group element. Then it does the following:

- If $\psi = \psi^*$ (which can only happen if the encryption oracle has not yet been invoked), then it simply halts (this corresponds to the early-abort rule introduced in game $\mathbf{G}_1$); otherwise, it continues.
- It tests if $(a, b) \in \mathcal{V}_2$ for some $b \in G$.
- If this is so, then it finds the (unique) triple in $\mathcal{V}_1$ of the form (a, b, K) for some K and returns this value of K as the result of the decryption oracle invocation; otherwise, it continues.
- It tests if $(a, K) \in \mathcal{V}_3$ for some K.
- If this is so, then it returns this value of K as the result of the decryption oracle; otherwise, it generates a random bit string K of length $\mathsf{KDF.OutLen}(\lambda)$, adds the pair (a, K) to $\mathcal{V}_3$, and returns this value of K as the result of the decryption oracle invocation.

It is straightforward to verify by inspection that A_1 as above does the job.

That completes the proof of Theorem 10.2.

10.5. The security of CS3b in the random oracle model. We can now prove the following security theorem for CS3b in the random oracle model.

THEOREM 10.3. *The scheme* CS3b *is secure in the random oracle model if the CDH assumption holds for* $\mathcal{G}$ *and the TCR assumption holds for* HF.

Proof. To prove this, let us assume by way of contradiction that the CDH assumption holds for $\mathcal{G}$ and the TCR assumption holds for HF, but CS3b is not secure in the random oracle model.

Now, the CDH assumption implies that for any polynomials P_1 and P_2 (with integer coefficients, taking positive values on $\mathbf{Z}_{\geq 0}$) there exists a $\lambda_0 \in \mathbf{Z}_{\geq 0}$ such that for all $\lambda \geq \lambda_0$

$$\Pr[q \leq P_1(\lambda) : \Gamma[\hat{G}, G, g, q] \stackrel{R}{\leftarrow} \mathbf{S}_\lambda] \leq 1/P_2(\lambda),$$

since otherwise a trivial, brute-force algorithm would have a CDH advantage that was not negligible. This implies, in particular, that when we model KDF as a random oracle, it acts as a secure key derivation scheme. From this it follows from Theorems 9.1 and 9.2 that CS3b is secure in the random oracle model if the DDH assumption holds; actually, since these two theorems do not deal with the random oracle model, one must make a cursory inspection of the proofs of these theorems to draw this conclusion, but this is very straightforward.

Let A be a polynomial-time adversary that breaks the security of CS3b in the random oracle model. This means that there exist polynomials P_1, P_2 (with integer coefficients, taking positive values on $\mathbf{Z}_{\geq 0}$), an infinite set $\Lambda \subset \mathbf{Z}_{\geq 0}$, and sets $\mathcal{Z}_\lambda \subset [\mathbf{S}_\lambda]$ for each $\lambda \in \Lambda$ such that

- for all $\lambda \in \Lambda$ and $\Gamma \in \mathcal{Z}_\lambda$, $\mathsf{AdvCCA}_{\mathsf{CS3b},\mathsf{A}}(\lambda \mid \Gamma) \geq 1/P_1(\lambda)$;
- for all $\lambda \in \Lambda$, $\Pr_{\mathbf{S}_\lambda}[\mathcal{Z}_\lambda] \geq 1/P_2(\lambda)$.

Theorems 9.1 and 9.2 (adapted to the random oracle model), together with our TCR assumption, imply that there exists a polynomial-time algorithm A_1 such that

for all sufficiently large $\lambda \in \Lambda$ and for all but a negligible fraction of Γ in $\mathcal{Z}_\lambda$ we have

$$\mathsf{AdvDDH}_{\mathcal{G},\mathsf{A}_1}(\lambda \mid \Gamma) \geq 1/(2P_1(\lambda)).$$

We now apply Lemma 4.3 using the above algorithm A_1 and choosing the value of κ in that lemma so that $2^{-\kappa} \cdot Q'_{\mathsf{A}}(\lambda) \leq 1/2$, yielding a polynomial-time algorithm A_2 such that for all sufficiently large $\lambda \in \Lambda$, for all but a negligible fraction of $\Gamma \in \mathcal{Z}_\lambda$, and for all $\rho \in \mathcal{T}_{\lambda,\Gamma}$,

$$\Pr[\mathsf{A}_2(1^\lambda, \Gamma, \rho) \neq \mathsf{DHP}_{\lambda,\Gamma}(\rho)] \leq 1/(2Q'_{\mathsf{A}}(\lambda)).$$

Applying Theorem 10.1 with the adversary A yields a polynomial-time adversary A_3 such that for all $\lambda \in \Lambda$ and $\Gamma \in \mathcal{Z}_\lambda$, $\mathsf{AdvCCA}_{\mathsf{HEG},\mathsf{A}_3}(\lambda \mid \Gamma) \geq 1/P_1(\lambda)$. Applying Theorem 10.2 with the adversary A_3 yields a polynomial-time oracle machine A_4 such that

$$\mathsf{AdvCDH}^*_{\mathcal{G},\mathsf{A}_4}(\lambda \mid \Gamma) \geq 1/(2P_1(\lambda))$$

for all sufficiently large $\lambda \in \Lambda$ and for all but a negligible fraction of $\Gamma \in \mathcal{Z}_\lambda$. Since for a given value of λ algorithm A_4 makes no more than $Q'_{\mathsf{A}}(\lambda)$ DDH-oracle queries, if we replace the DDH-oracle used by A_4 with algorithm A_2 above, we obtain a polynomial-time algorithm A_5 such that for all sufficiently large $\lambda \in \Lambda$ and for all but a negligible fraction of Γ in $\mathcal{Z}_\lambda$ we have $\mathsf{AdvCDH}_{\mathcal{G},\mathsf{A}_5}(\lambda \mid \Gamma) \geq 1/(4P_1(\lambda))$. But this contradicts the CDH assumption. □

10.6. Random oracles and pairwise independent key derivation functions: Getting the best of both. If we want to prove the security of CS3b in the standard model without making any intractability assumptions about KDF, then we may choose KDF to be pairwise independent. On the one hand, standard constructions for pairwise independent hash functions typically exhibit a lot of algebraic structure, and it is not very reasonable to assume that such a KDF can be safely modeled as a random oracle. On the other hand, typical dedicated cryptographic hash functions, such as SHA-1, may be modeled as random oracles, but they are certainly not pairwise independent.

We shall sketch here how to get the best of both worlds, i.e., how to implement the KDF so that we get a proof of security of CS3b in the standard model just under the DDH and TCR assumptions and in the random oracle model under the CDH and TCR assumptions.

The idea is this: compute KDF as the XOR of a pairwise independent hash KDF1 and a cryptographic hash KDF2.

It is clear that if KDF1 is pairwise independent, then so is KDF, and so the security of CS3b in the standard model under the DDH and TCR assumptions now follows directly from Theorem 9.2.

Now suppose we model the cryptographic hash KDF2 as a random oracle. It is easy to see that for any adversary A attacking CS3b given oracle access to KDF2 there is an adversary A_1, whose running time is roughly the same as that of A, that attacks CS3b given oracle access to KDF: the adversary A_1 just does whatever A does, except that whenever A queries the oracle for KDF2 adversary A_1 queries its oracle for KDF and computes the value of KDF2 as the XOR of the value of KDF and the value of KDF1. Note, however, that the output distribution of the oracle KDF is the same as that of a random oracle, and so the security of CS3b in the random oracle model under the CDH and TCR assumptions now follows directly from Theorem 10.3.

We do not necessarily advocate this approach to building a KDF in practical implementations: simply assuming that a KDF implemented directly using a dedicated cryptographic hash is secure is quite reasonable, and the resulting KDF is much simpler and more efficient than any approach that makes use of a pairwise independent hash function.

10.7. Further discussion. The scheme HEG is intended to represent a fairly traditional version of ElGamal key encapsulation. The only thing slightly nontraditional about it is the fact that the symmetric key K is derived by hashing both a (the ephemeral Diffie–Hellman public key) and b (the shared Diffie–Hellman key), rather than just b alone.

Hashing both the ephemeral and shared keys together has some quantitative security advantages. Notice that in Theorem 10.2 the implied CDH algorithm makes no more than $Q'_{\mathsf{A}}(\lambda)$ queries to the DDH-oracle. If we were to hash only the shared Diffie–Hellman key, we could still prove the security of HEG, but the reduction would be less efficient; in particular, the implied CDH algorithm might require up to $Q'_{\mathsf{A}}(\lambda) \cdot Q_{\mathsf{A}}(\lambda)$ queries to the DDH-oracle. A similar quantitative security advantage arises in the multiuser/multimessage model (see [7]). In this model, we can exploit the well-known random self-reducibility of the CDH problem to get a more efficient reduction if we hash both keys instead of just one. Of course, these improved security reductions for HEG carry over to the security reduction for CS3b in the random oracle model.

The DHAES encryption scheme [1], which is a hybrid ElGamal encryption scheme that has been proposed for standardization, also hashes both the ephemeral and shared Diffie–Hellman keys to derive a symmetric key. Indeed, the DHAES scheme can be constructed from the key encapsulation mechanism HEG using the hybrid constructions presented in section 7, and it is straightforward to verify that analogues of Theorems 10.1 and 10.2 hold for the DHAES scheme as well. The DHAES scheme needs to hash both group elements because it allows the possibility of a group G whose order is a composite number. In a revised version of DHAES, called DHIES [2], the group G is required to have prime order, and only the shared Diffie–Hellman key is hashed. However, as we have seen, there are still some security benefits to be gained from hashing both group elements, even if the group is of prime order, as we are assuming in this paper.

We should also point out that because any attack on scheme CS3b can be immediately translated into an attack on HEG (see Theorem 10.1), it seems very unlikely that any of the side channels discussed in Remark 9.4 could be used to break CS3b without breaking HEG. Thus, although the availability of such side channel information would invalidate the proof of security of CS3b under the DDH, it seems very unlikely that it could be exploited to break it.

Theorem 10.3 originally appeared in the paper [56]. The proof in that paper basically rolled all of the arguments used in the proofs of Theorems 10.1, 10.2, and 10.3, along with the arguments in section 10.6, into a single proof, which we have unraveled to some extent here. Our presentation here was somewhat influenced by the paper [48], which formally introduces the notion of the CDH assumption relative to an oracle for the DDH problem.

The security reduction in Theorem 10.3 is quite inefficient: we have to perform many simulations using the given adversary A just to solve one instance of the DDH problem, and then in a different simulation involving A we have to solve many instances of the DDH problem in order to solve one instance of the CDH problem. Of course, if the DDH problem for a given group scheme turns out not to be a hard

problem, then it may very well be the case that there is a much more efficient DDH algorithm than the one built using our security reduction involving A. In this case, the reduction in Theorem 10.3 becomes quite reasonable.

11. Summary. We conclude with a brief summary of the schemes presented in this paper:

- the public-key encryption scheme CS1, which is secure under the DDH and TCR assumptions, along with minor, slightly simpler, and more efficient variants CS1a and CS1b;
- the public-key encryption scheme CS2, which is a "hash free" variant of CS1 and which is secure under the DDH assumption;
- the key encapsulation mechanism CS3, along with variants CS3a and CS3b, which can be combined with an appropriate symmetric cipher to build a hybrid cipher; the hybrid cipher will be secure under the DDH and TCR assumptions, together with an appropriate "smoothing" assumption for the key derivation function, and an appropriate security assumption for the symmetric cipher.

Among these schemes, the one most likely to be used in practice is the hybrid construction based on CS3b. Over the other schemes, it has the following advantages:

- it is more efficient;
- it can be used to encrypt messages that are bit strings of arbitrary length;
- it is no less secure than traditional "hashed" ElGamal;
- it can be proven secure in the random oracle model under weaker CDH assumption.

A version of this scheme is currently included in a draft ISO standard for public key encryption.

Acknowledgments. Some of this work was done while the first author was at the Institute for Theoretical Computer Science, ETH, Zurich, and while the second author was at the IBM Zurich Research Lab, as well as visiting the Computer Science Department at Stanford University. Thanks to Ilia Mironov for comments on an early draft of this paper.

Thanks also to Moni Naor for his comments on an early draft of the *Crypto* '98 paper, on which this paper is based, and, in particular, for his suggestion of using a universal one-way hash function instead of a collision resistant hash function in the design of scheme CS1 and for his suggestion of a hash-free variant, upon which scheme CS2 is loosely based.

REFERENCES

[1] M. Abdalla, M. Bellare, and P. Rogaway, *DHAES: An Encryption Scheme Based on the Diffie-Hellman Problem*, Cryptology ePrint Archive, Report 1999/007, http://eprint.iacr.org (1999).

[2] M. Abdalla, M. Bellare, and P. Rogaway, *The oracle Diffie-Hellman assumptions and an analysis of DHIES*, in Topics in Cryptology—CT-RSA 2001, Lecture Notes in Comput. Sci. 2020, D. Naccache, ed., Springer-Verlag, Heidelberg, 2001, pp. 143–158.

[3] N. Asokan, V. Shoup, and M. Waidner, *Optimistic fair exchange of digital signatures*, IEEE Journal on Selected Areas in Communications, 18 (2000), pp. 593–610.

[4] E. Bach and J. Shallit, *Algorithmic Number Theory*, Vol. 1, MIT Press, Cambridge, MA, 1996.

[5] P. Bateman and R. Horn, *A heuristic asymptotic formula concerning the distribution of prime numbers*, Math. Comp., 16 (1962), pp. 363–367.

[6] P. Bateman and R. Horn, *Primes represented by irreducible polynomials in one variable*, in Theory of Numbers, Proc. Sympos. Pure Math. 8, A. L. Whiteman, ed., AMS, Providence, RI, 1965, pp. 119–132.

[7] M. Bellare, A. Boldyreva, and S. Micali, *Public-key encryption in a multi-user setting: Security proofs and improvements*, in Advances in Cryptology—Eurocrypt 2000, Lecture Notes in Comput. Sci. 1807, B. Preneel, ed., Springer-Verlag, Heidelberg, 2000, pp. 259–274.

[8] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, *Relations among notions of security for public-key encryption schemes*, in Advances in Cryptology—Crypto '98, Lecture Notes in Comput. Sci. 1462, H. Krawczyk, ed., Springer-Verlag, Heidelberg, 1998, pp. 26–45.

[9] M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, in Proceedings of the First ACM Conference on Computer and Communications Security, Fairfax, VA, 1993, pp. 62–73.

[10] M. Bellare and P. Rogaway, *Optimal asymmetric encryption*, in Advances in Cryptology—Eurocrypt '94, Lecture Notes in Comput. Sci. 950, A. De Santis, ed., Springer-Verlag, Heidelberg, 1994, pp. 92–111.

[11] M. Bellare and P. Rogaway, *Collision-resistant hashing: Towards making UOWHFs practical*, in Advances in Cryptology—Crypto '97, Lecture Notes in Comput. Sci. 1294, B. S. Kaliski, Jr., ed., Springer-Verlag, Heidelberg, 1997, pp. 470–484.

[12] I. Blake, G. Seroussi, and N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, Cambridge, UK, 1999.

[13] D. Bleichenbacher, *Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1*, in Advances in Cryptology—Crypto '98, Lecture Notes in Comput. Sci. 1462, H. Krawczyk, ed., Springer-Verlag, Heidelberg, 1998, pp. 1–12.

[14] D. Boneh, *The decision Diffie-Hellman problem*, in ANTS-III, Lecture Notes in Comput. Sci. 1423, J. P. Buhler, ed., Springer-Verlag, Berlin, 1998, pp. 48–63.

[15] D. Boneh, *Simplified OAEP for the RSA and Rabin functions*, in Advances in Cryptology—Crypto 2001, Lecture Notes in Comput. Sci. 2139, J. Kilian, ed., Springer-Verlag, Heidelberg, 2001, pp. 275–291.

[16] S. Brands, *An Efficient Off-Line Electronic Cash System Based on the Representation Problem*, CWI Technical report CS-R9323, Centre for Mathematics and Computer Science, Amsterdam, 1993.

[17] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson, *Fast exponentiation with precomputation*, in Advances in Cryptology—Eurocrypt '92, Lecture Notes in Comput. Sci. 658, R. A. Rueppel, ed., Springer-Verlag, Heidelberg, 1993, pp. 200–207.

[18] R. Canetti, *Universally Composable Security: A New Paradigm for Cryptographic Protocols*, Cryptology ePrint Archive, Report 2000/067, http://eprint.iacr.org (2000).

[19] R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology, revisited*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 209–218.

[20] R. Canetti and S. Goldwasser, *An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack*, in Advances in Cryptology—Eurocrypt '99, Lecture Notes in Comput. Sci. 1592, J. Stern, ed., Springer-Verlag, Heidelberg, 1999, pp. 90–106.

[21] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, MA, 2002.

[22] R. Cramer and V. Shoup, *A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack*, in Advances in Cryptology—Crypto '98, Lecture Notes in Comput. Sci. 1462, H. Krawczyk, ed., Springer-Verlag, Heidelberg, 1998, pp. 13–25.

[23] R. Cramer and V. Shoup, *Signature schemes based on the strong RSA assumption*, ACM Transactions on Information and Systems Security, 3 (2000), pp. 161–185.

[24] R. Cramer and V. Shoup, *Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public key encryption*, in Advances in Cryptology—Eurocrypt 2002, Lecture Notes in Comput. Sci. 2332, L. R. Knudsen, ed., Springer-Verlag, Heidelberg, 2002, pp. 45–64.

[25] I. Damgård, *Towards practical public key cryptosystems secure against chosen ciphertext attacks*, in Advances in Cryptology—Crypto '91, Lecture Notes in Comput. Sci. 576, J. Feigenbaum, ed., Springer-Verlag, Heidelberg, 1992, pp. 445–456.

[26] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Trans. Information Theory, 22 (1976), pp. 644–654.

[27] D. Dolev, C. Dwork, and M. Naor, *Non-malleable cryptography*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, New Orleans, LA, 1991, pp. 542–552.

[28] D. Dolev, C. Dwork, and M. Naor, *Nonmalleable cryptography*, SIAM J. Comput., 30 (2000), pp. 391–437.
[29] C. Dwork and M. Naor, *Method for Message Authentication from Non-malleable Cryptosystems*, U.S. Patent No. 05539826, 1996.
[30] T. ElGamal, *A public key cryptosystem and signature scheme based on discrete logarithms*, IEEE Trans. Inform. Theory, 31 (1985), pp. 469–472.
[31] Y. Frankel and M. Yung, *Cryptanalysis of immunized LL public key systems*, in Advances in Cryptology—Crypto '95, Lecture Notes in Comput. Sci. 963, D. Coppersmith, ed., Springer-Verlag, Heidelberg, 1995, pp. 287–296.
[32] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, *RSA-OAEP is secure under the RSA assumption*, in Advances in Cryptology—Crypto 2001, Lecture Notes in Comput. Sci. 2139, J. Kilian, ed., Springer-Verlag, Heidelberg, 2001, pp. 260–274.
[33] O. Goldreich and L. A. Levin, *A hard-core predicate for all one-way functions*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 25–32.
[34] S. Goldwasser and S. Micali, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.
[35] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.
[36] R. Impagliazzo, L. Levin, and M. Luby, *Pseudo-random number generation from any one-way function*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 12–24.
[37] R. Impagliazzo and D. Zuckermann, *How to recycle random bits*, in Proceedings of the 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, NC, 1989, pp. 248–253.
[38] A. Joux and K. Nguyen, *Separating Decision Diffie-Hellman from Diffie-Hellman in Cryptographic Groups*, Cryptology ePrint Archive, Report 2001/003, http://eprint.iacr.org (2001).
[39] C. H. Lim and P. J. Lee, *Another method for attaining security against adaptively chosen ciphertext attacks*, in Advances in Cryptology—Crypto '93, Lecture Notes in Comput. Sci. 773, D. R. Stinson, ed., Springer-Verlag, Heidelberg, 1994, pp. 420–434.
[40] C. H. Lim and P. J. Lee, *More flexible exponentiation with precomputation*, in Advances in Cryptology—Crypto '94, Lecture Notes in Comput. Sci. 839, Y. G. Desmedt, ed., Springer-Verlag, Heidelberg, 1994, pp. 95–107.
[41] J. Manger, *A chosen ciphertext attack on RSA optimal asymmetric encryption padding (OAEP) as standardized in PKCS # 1 v2.0*, in Advances in Cryptology—Crypto 2001, Lecture Notes in Comput. Sci. 2139, J. Kilian, ed., Springer-Verlag, Heidelberg, 2001, pp. 230–238.
[42] U. Maurer and S. Wolf, *Diffie-Hellman, decision Diffie-Hellman, and discrete logarithms*, in Proceedings of the IEEE International Symposium on Information Theory (ISIT '98), Cambridge, MA, 1998, p. 327.
[43] U. Maurer and S. Wolf, *The Diffie-Hellman protocol*, Des. Codes Cryptogr., 19 (2000), pp. 147–171.
[44] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, 1997.
[45] M. Naor and O. Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 458–467.
[46] M. Naor and M. Yung, *Universal one-way hash functions and their cryptographic applications*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 33–43.
[47] M. Naor and M. Yung, *Public-key cryptosystems provably secure against chosen ciphertext attacks*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, MD, 1990, pp. 427–437.
[48] T. Okamoto and D. Pointcheval, *The gap-problems: A new class of problems for the security of cryptographic schemes*, in Proceedings of the 2001 International Workshop on Practice and Theory in Public Key Cryptography (PKC 2001), Cheju Island, Korea, 2001.
[49] P. Paillier, *Public-key cryptosystems based on composite degree residuosity classes*, in Advances in Cryptology—Eurocrypt '99, Lecture Notes in Comput. Sci. 1592, J. Stern, ed., Springer-Verlag, Heidelberg, 1999, pp. 223–238.
[50] C. Rackoff and D. Simon, *Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack*, in Advances in Cryptology—Crypto '91, Lecture Notes in Comput. Sci. 576, J. Feigenbaum, ed., Springer-Verlag, Heidelberg, 1992, pp. 433–444.

[51] J. Rompel, *One-way functions are necessary and sufficient for digital signatures*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, MD, 1990, pp. 387–394.

[52] *Secure Hash Standard*, FIPS Publication 180-1, National Institute of Standards and Technology (NIST), Gaithersburg, MD, 1995.

[53] V. Shoup, *Lower bounds for discrete logarithms and related problems*, in Advances in Cryptology—Eurocrypt '97, Lecture Notes in Comput. Sci. 1233, W. Fumy, ed., Springer-Verlag, Heidelberg, 1997, pp. 256–266.

[54] V. Shoup, *On Formal Models for Secure Key Exchange*, Cryptology ePrint Archive, Report 1999/012, http://eprint.iacr.org (1999).

[55] V. Shoup, *A composition theorem for universal one-way hash functions*, in Advances in Cryptology—Eurocrypt 2000, Lecture Notes in Comput. Sci. 1807, B. Preneel, ed., Springer-Verlag, Heidelberg, 2000, pp. 445–452.

[56] V. Shoup, *Using hash functions as a hedge against chosen ciphertext attack*, in Advances in Cryptology—Eurocrypt 2000, Lecture Notes in Comput. Sci. 1807, B. Preneel, ed., Springer-Verlag, Heidelberg, 2000, pp. 275–288.

[57] V. Shoup, *OAEP reconsidered*, in Advances in Cryptology—Crypto 2001, Lecture Notes in Comput. Sci. 2139, J. Kilian, ed., Springer-Verlag, Heidelberg, 2001, pp. 239–259.

[58] V. Shoup and R. Gennaro, *Securing threshold cryptosystems against chosen ciphertext attack*, J. Cryptology, 5 (2002), pp. 75–96.

[59] D. Simon, *Finding collisions on a one-way street: Can secure hash functions be based on general assumptions?*, in Advances in Cryptology—Eurocrypt '98, Lecture Notes in Comput. Sci. 1403, G. Goos and K. Nyberg, eds., Springer-Verlag, Heidelberg, 1998, pp. 334–345.

[60] N. Smart, *The discrete logarithm problem on elliptic curves of trace one*, J. Cryptology, 12 (1999), pp. 193–196.

[61] M. Stadler, *Publicly verifiable secret sharing*, in Advances in Cryptology—Eurocrypt '96, Lecture Notes in Comput. Sci. 1070, U. Maurer, ed., Springer-Verlag, Heidelberg, 1996, pp. 190–199.

[62] Y. Zheng and J. Seberry, *Practical approaches to attaining security against adaptively chosen ciphertext attacks*, in Advances in Cryptology—Crypto '92, Lecture Notes in Comput. Sci. 740, E. F. Brickell, ed., Springer-Verlag, Heidelberg, 1993, pp. 292–304.

SIAM J. COMPUT.
Vol. 33, No. 1, pp. 227–260

COUNTING COMPLEXITY CLASSES FOR NUMERIC COMPUTATIONS I: SEMILINEAR SETS*

PETER BÜRGISSER† AND FELIPE CUCKER‡

Abstract. We define a counting class $\#\mathrm{P}_{\mathrm{add}}$ in the Blum–Shub–Smale setting of additive computations over the reals. Structural properties of this class are studied, including a characterization in terms of the classical counting class $\#\mathsf{P}$ introduced by Valiant. We also establish transfer theorems for both directions between the real additive and the discrete setting. Then we characterize in terms of completeness results the complexity of computing basic topological invariants of semilinear sets given by additive circuits. It turns out that the computation of the Euler characteristic is $\mathrm{FP}_{\mathrm{add}}^{\#\mathrm{P}_{\mathrm{add}}}$-complete, while for fixed k the computation of the kth Betti number is $\mathrm{FPAR}_{\mathrm{add}}$-complete. Thus the latter is more difficult under standard complexity theoretic assumptions. We use all of the above to prove some analogous completeness results in the classical setting.

Key words. counting complexity, real complexity classes, Euler characteristic, Betti numbers

AMS subject classifications. 68Q15, 68Q17, 55-04

DOI. 10.1137/S0097539702415950

1. Introduction.

In 1989 Blum, Shub, and Smale [5] introduced a theory of computation over the real numbers with the goal of providing numerical computations (as performed, e.g., in numerical analysis or computational geometry) the kind of foundations classical complexity theory has provided to discrete computation. This theory describes the difficulty of solving numerical problems and provides a taxonomy of complexity classes capturing different degrees of such a difficulty.

Since its introduction, this Blum–Shub–Smale (BSS) theory has focused mainly on decisional problems. Functional problems attracted attention at the level of analysis of particular algorithms, but structural properties of classes of such problems were hardly studied. So far, the only systematic approach to study the complexity of certain functional problems within a framework of computations over the reals is Valiant's theory of VNP-completeness [7, 40, 43]. However, the relationship of this theory to the more general BSS setting is, as of today, poorly understood. A detailed account of the research on complexity of real functions within the classical framework can be found in [23].

A first step in the study of functional properties could focus on complexity classes related to counting problems, i.e., functional problems, whose associated functions count the number of solutions of some decisional problem.

In classical complexity theory, counting classes were introduced by Valiant in his seminal papers [41, 42]. Valiant defined $\#\mathsf{P}$ as the class of functions which count the number of accepting paths of nondeterministic polynomial time machines and proved that the computation of the permanent is $\#\mathsf{P}$-complete. This exhibited an unexpected difficulty for the computation of a function, whose definition is only slightly different from that of the determinant, a problem known to be in $\mathsf{FNC}^2 \subseteq \mathsf{FP}$, and thus

*Received by the editors October 10, 2002; accepted for publication (in revised form) September 19, 2003; published electronically December 31, 2003. This work was supported by SRG grant 7001558 and DFG grant BU 1371.

http://www.siam.org/journals/sicomp/33-1/41595.html

†Faculty of Computer Science, Electrical Engineering and Mathematics, Paderborn University, D-33095 Paderborn, Germany (pbuerg@upb.de).

‡Department of Mathematics, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong (macucker@math.cityu.edu.hk).

considered "easy." This difficulty was highlighted by a result of Toda [38] proving that $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$, i.e., that $\#\mathsf{P}$ has at least the power of the polynomial hierarchy.

In the continuous setting, i.e., over the reals, the only attempt to define counting classes was made by Meer [27]. He defined a real version of the class $\#\mathsf{P}$ and studied some of its logical properties in terms of metafinite model theory. Meer did not investigate complete problems for this class.

In this paper we will define and study counting classes in the model of additive BSS machines [24]. The computation nodes of these machines perform additions and subtractions but no multiplications and divisions. The corresponding complexity classes are denoted by $\mathrm{P}_{\mathrm{add}}$ and $\mathrm{NP}_{\mathrm{add}}$.[1]

The results in this paper can be seen as a first step towards a better understanding of the power of counting in the unrestricted BSS model over the reals (allowing also for multiplications and divisions). A sequel to this paper studying this setting is under preparation [11].

Our results can be grouped in two kinds: structural relationships between complexity classes and completeness results. The latter (for whose proofs the former are used) satisfy a driving motivation for this paper: to capture the complexity of computing basic topological invariants of geometric objects in terms of complexity classes and completeness results. In the following, we give an outline of the main results of this paper.

1.1. Counting classes. Recall that $\#\mathsf{P}$ is the class of functions $f\colon \{0,1\}^\infty \to \mathbb{N}$ for which there exists a polynomial time Turing machine M and a polynomial p with the property that for all $n \in \mathbb{N}$ and all $x \in \{0,1\}^n$, $f(x)$ counts the number of strings $y \in \{0,1\}^{p(n)}$ such that M accepts (x, y).

By replacing Turing machines with additive BSS machines in the above definition, we get a class of functions $f : \mathbb{R}^\infty \to \mathbb{N} \cup \{\infty\}$, which we denote by $\#\mathrm{P}_{\mathrm{add}}$. Thus $f(x)$ counts the number of vectors $y \in \mathbb{R}^{p(n)}$ such that M accepts (x, y). By counting only the number of "digital" vectors $y \in \{0,1\}^{p(n)}$, we obtain a smaller class of functions $f : \mathbb{R}^\infty \to \mathbb{N}$ denoted by $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$.

In Theorem 4.7 we show that a counting problem $f \in \mathrm{D}\#\mathrm{P}_{\mathrm{add}}$ is $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$-complete with respect to Turing reductions iff it is $\#\mathrm{P}_{\mathrm{add}}$-complete with respect to Turing reductions. Moreover, in section 4.3 we prove that there is a wealth of natural complete problems for the class $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$ with respect to Turing reductions. For instance, consider the following counting version of the real weighted perfect matching problem $\#\mathrm{PM}_{\mathbb{R}}$: given $w \in \mathbb{R}$ and a bipartite graph G with real weights on the edges, count the number of perfect matchings of G having weight at most w. (The weight of a matching is defined as the sum of the weights of its edges.) The problem $\#\mathrm{PM}_{\mathbb{R}}$ turns out to be $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$-complete with respect to Turing reductions. The same is true for the counting version of the real traveling salesman problem $\#\mathrm{TSP}_{\mathbb{R}}$ to count the number of Hamilton circuits of weight at most w of a given graph with real weights on the edges. It is an interesting open problem whether these problems are also $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$-complete with respect to parsimonious reductions; see section 7.

The above completeness results follow from a general principle (Proposition 4.13), which says that for proving $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$-completeness of the counting version of a problem in $\mathrm{DNP}_{\mathrm{add}}$ it is sufficient to show that the restriction of the corresponding counting problem to integer inputs is $\#\mathsf{P}$-complete. The proof of this principle is based on an

[1] To distinguish between classical and additive complexity complexity classes, we use the subscript "add" to indicate the latter. Also, to further emphasize this distinction, we write the former in sans serif.

extension of the structural relationships between the real additive and the discrete setting, discovered by Fournier and Koiran [16, 17], discussed next.

1.2. Structural relationships. The main result of section 4.1 is summarized in Theorem 4.1, which says that for several classical[2] complexity classes $\mathcal{C}$ consisting of decisional problems the corresponding additive complexity class $\mathcal{C}_{\mathrm{add}}$ is contained in, or even equal to, $\mathrm{P}^{\mathcal{C}}_{\mathrm{add}}$. That is, all problems in $\mathcal{C}_{\mathrm{add}}$ can be solved by an additive machine working in polynomial time and having access to a (discrete) oracle in $\mathcal{C}$. Likewise, if $\mathcal{C}$ is a classical complexity class of functions $\{0,1\}^\infty \to \{0,1\}^\infty$, we obtain that $\mathcal{C}_{\mathrm{add}} \subseteq \mathrm{FP}^{\mathcal{C}}_{\mathrm{add}}$. In particular, we have $\mathrm{FP}^{\#\mathrm{P}_{\mathrm{add}}}_{\mathrm{add}} = \mathrm{FP}^{\#\mathsf{P}}_{\mathrm{add}}$.

Theorem 4.1 is an extension of the work of Fournier and Koiran [16, 17], who discovered this close relationship between the real additive and the discrete setting. This relationship is based on Meyer auf der Heide's (nonuniform) construction of small depth linear decision trees for point location in arrangements of hyperplanes [29, 30] (see also Meiser [28] for an extension of these results). Fournier and Koiran showed that this construction can be made uniform if a classical NP-oracle is available. This way, they proved that $\mathrm{NP}_{\mathrm{add}} \subseteq \mathrm{P}^{\mathsf{NP}}_{\mathrm{add}}$. We have extended this result in various directions, in particular to the counting context.

An interesting application of the above insights is that Toda's famous result [38], as well as its extension by Toda and Watanabe [39], carry over to the real additive setting (Corollary 4.6). We use this to prove that the counting class $\#\mathrm{P}_{\mathrm{add}}$ is closely related to its digital variant $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$ in the sense that $\mathrm{FP}^{\#\mathrm{P}_{\mathrm{add}}}_{\mathrm{add}} = \mathrm{FP}^{\mathrm{D}\#\mathrm{P}_{\mathrm{add}}}_{\mathrm{add}}$. In other words, a $\#\mathrm{P}_{\mathrm{add}}$-oracle does not give more power to an additive polynomial time Turing machine than a $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$-oracle.

An important application of our structural insights is the following transfer result (Corollary 4.11):

$$\#\mathrm{P}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}} \Longleftrightarrow \mathrm{D}\#\mathrm{P}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}} \Longleftrightarrow \#\mathsf{P} \subseteq \mathsf{FP/poly}.$$

The proof uses the fact that the Boolean part of $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$, consisting of the restrictions of all functions in $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$ to the set of binary inputs $\{0,1\}^\infty$, is equal to $\#\mathsf{P/poly}$ (Proposition 4.10).

1.3. Topological invariants. Algebraic topology studies topological spaces X by assigning to X various algebraic objects in a functorial way. In particular, homeomorphic (or even homotopy equivalent) spaces lead to isomorphic algebraic objects. For a general reference in algebraic topology we refer the reader to [20, 33]. Typical examples of such algebraic objects studied are the (singular) homology vector spaces $H_k(X;\mathbb{Q})$ over $\mathbb{Q}$, defined for integers $k \in \mathbb{N}$. The dimension $b_k(X)$ of $H_k(X;\mathbb{Q})$ is called the kth *Betti number* of the space X. The zeroth Betti number $b_0(X)$ counts the number of connected components of X and for $k > 0$, $b_k(X)$, measures a more sophisticated "degree of connectivity." Intuitively speaking, for a three-dimensional space X, $b_1(X)$ counts the number of holes and $b_2(X)$ counts the number of cavities of X. It is known that $b_k(X) = 0$ for $k > n := \dim X$. The *Euler characteristic* of X defined by $\chi(X) := \sum_{k=0}^{n} (-1)^k b_k(X)$ is an important numerical invariant of X, enjoying several nice properties. For a finite set X, $\chi(X)$ is just the cardinality of X.

The notion of a cell complex [20, 33] will be of importance for our algorithms to compute the Euler characteristic and the Betti numbers. For instance, if X is

[2]Throughout this paper we use the words *discrete*, *classical*, or *Boolean* to emphasize that we are referring to the theory of complexity over a finite alphabet as exposed, e.g., in [2, 34].

decomposed as a finite cell complex having c_k cells of dimension k, then $\chi(X) := \sum_{k=0}^{n}(-1)^k c_k$.

We remark that the number of connected components, the Euler characteristic, and the Betti numbers lead to interesting lower complexity bounds for semialgebraic decision problems; see [3, 44, 45] and the survey [9].

1.4. Semilinear sets and additive circuits. In this paper, we will confine our investigations to *semilinear sets* $X \subseteq \mathbb{R}^n$, which are derived from closed halfspaces by taking a finite number of unions, intersections, and complements. Moreover, we assume that the closed halfspaces are given by linear inequalities of "mixed type" $a_1X_1 + \cdots + a_nX_n \leq b$ with integer coefficients a_i and real right-hand side b.

We will represent semilinear sets by a very compact data structure. An *additive circuit* $\mathcal{C}$ is a special arithmetic circuit [19], whose set of arithmetic operations is restricted to additions and subtractions. The circuit may have selection gates and use a finite set of real constants. (See Definition 2.9 for details.) The set of inputs accepted by an additive circuit is semilinear, and any semilinear set can be described this way.

The basic problem $\mathrm{CSAT}_{\mathrm{add}}$ to decide whether the semilinear set X given by an additive circuit is nonempty turns out to be $\mathsf{NP}_{\mathrm{add}}$-complete [4]. By contrast, the feasibility question for a system of linear inequalities of the above mixed type is solvable in $\mathrm{P}_{\mathrm{add}}$. This is just a rephrasing of a well-known result by Tardos [37] (cf. Remark 2.7).

Over the real numbers, space is not as meaningful a resource as it is in the discrete setting (cf. [31]). The role of space, however, is satisfactorily played by parallel time formalized by the notion of uniform arithmetic circuits (cf. [4, 14]). We denote by $\mathrm{PAR}_{\mathrm{add}}$ the class of decision problems for which there exists a $\mathrm{P}_{\mathrm{add}}$-uniform family $(\mathcal{C}_n)$ of additive circuits such that the depth of $\mathcal{C}_n$ grows at most polynomially in n (see section 2). $\mathrm{FPAR}_{\mathrm{add}}$ denotes the class of functions f which can be computed with such resources and such that the size of $f(x)$ is polynomially bounded in the size of x. (The size of a vector is defined as its length.)

1.5. Completeness results for topological invariants. In the computational problems listed below, it is always assumed that the input is an additive circuit $\mathcal{C}$ and X is the semilinear set accepted by $\mathcal{C}$. We also say that X is defined or given by $\mathcal{C}$.

Among the completeness results proved in this paper, the most important ones are the following (for a complete list see section 6).

1. The problem $\mathrm{DIM}_{\mathrm{add}}(d)$ to decide whether $\dim X \geq d$ is $\mathrm{NP}_{\mathrm{add}}$-complete (Theorem 5.1).

2. The problem $\mathrm{EULER}_{\mathrm{add}}$ to compute the Euler characteristic of a closed semilinear set X is $\mathrm{FP}_{\mathrm{add}}^{\#\mathrm{P}_{\mathrm{add}}}$-complete with respect to Turing reductions (Theorem 5.18).

3. The problem $\mathrm{BETTI}_{\mathrm{add}}(k)$ to compute the kth Betti number $b_k(X)$ of a closed semilinear set X is $\mathrm{FPAR}_{\mathrm{add}}$-complete with respect to Turing reductions (Theorem 5.19).

These results give a complexity theoretic distinction between the problems to compute the Euler characteristic and to compute Betti numbers. The computation of the Euler characteristic is strictly easier than the computation of the number of connected components, or more generally than the computation of the kth Betti number for any fixed k, under a standard complexity theoretic assumption (Corollary 5.23). Intuitively, the fact that $\mathrm{EULER}_{\mathrm{add}}$ is easier than $\mathrm{BETTI}_{\mathrm{add}}(k)$ can be explained by the various nice properties satisfied by the Euler characteristic.

Let us now restrict the inputs in the above three problems $\mathcal{P}$ to *constant-free* additive circuits and denote the resulting computational problem by $\mathcal{P}^0$. Note that constant-free circuits can be encoded over a finite alphabet and thus be handled by (classical) Turing machines. In section 5.3.6 we derive the following completeness results in the Turing model: $\mathrm{DIM}^0_{\mathrm{add}}(d)$ is NP-complete, $\mathrm{EULER}^0_{\mathrm{add}}$ is $\mathsf{FP}^{\#\mathsf{P}}$-complete, and $\mathrm{BETTI}^0_{\mathrm{add}}(k)$ is $\mathsf{FPSPACE}$-complete.

We briefly describe the proof idea for $\mathrm{BETTI}_{\mathrm{add}}(k)$. The lower bound is inspired by an early paper by Reif [35] (see also [36]), who showed the PSPACE-hardness of a generalized movers problem in robotics. The *reachability problem* $\mathrm{REACH}_{\mathrm{add}}$ is the following: given an additive circuit defining the semilinear set X and given two points $s, t \in X$, decide whether s and t are in the same connected component of X. Reif's result implies that the analogue of the reachability problem for semialgebraic sets (given by inequalities of rational polynomials) is PSPACE-hard. We cannot apply this result in our context, since we are dealing here with linear polynomials (of mixed type). However, we borrow from Reif's proof the idea to characterize PSPACE by symmetric polynomial space Turing machines [26] and prove that $\mathrm{REACH}_{\mathrm{add}}$ is $\mathrm{PAR}_{\mathrm{add}}$-hard (Proposition 5.9). From this lower bound result, one can derive the $\mathrm{PAR}_{\mathrm{add}}$-hardness of $\mathrm{BETTI}_{\mathrm{add}}(k)$ by standard constructions of algebraic topology.

The proof that $\mathrm{BETTI}_{\mathrm{add}}(k)$belongs to $\mathrm{FPAR}_{\mathrm{add}}$ proceeds by the following steps:

1. An additive circuit $\mathcal{C}$ accepting a set X defines a decomposition of X into leaf sets. This decomposition can be refined to a finite cell complex if X is compact (cf. section 5.3.3).
2. The matrices (a_{ij}) of the boundary maps of the corresponding cellular homology can be succinctly represented by Boolean circuits computing a_{ij} from the index pair (i, j) given in binary.
3. The rank of an integer matrix given in succinct representation can be computed in a space efficient manner (Lemma 5.21).

1.6. Organization of the paper. We start in section 2 by introducing some notation and recalling basic facts about additive machines. Then we define in section 3 the counting complexity class $\#\mathrm{P}_{\mathrm{add}}$ in the additive model as well as its digital variant $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$, introduce different notions of reductions, and prove some basic completeness results. Section 4 deals with structural relationships and can be seen as the first part of this paper. Section 5 about the complexity to compute topological invariants constitutes the second part of this paper. It contains completeness proofs for several natural computational problems, each of which are treated in separate subsections. Those problems are counting connected components, computing the Euler characteristic, and computing Betti numbers. We also present completeness results for the corresponding problems in the Turing model in section 5.3.6. Finally, we end the paper in section 6 with a summary of problems and results and with some selected open problems in section 7.

2. Preliminaries about additive machines. We denote by $\mathbb{R}^\infty$ the disjoint union $\mathbb{R}^\infty = \bigsqcup_{n\geq 0}\mathbb{R}^n$, where, for $n \geq 0$, $\mathbb{R}^n$ is the standard n-dimensional space over $\mathbb{R}$. The space $\mathbb{R}^\infty$ is a natural one to represent problem instances of arbitrarily high dimension. For $x \in \mathbb{R}^n \subset \mathbb{R}^\infty$, we call n the *size* of x, and we denote it by $\mathrm{size}(x)$. Contained in $\mathbb{R}^\infty$ is the set of bitstrings $\{0,1\}^\infty$ defined as the union of the sets $\{0,1\}^n$ for $n \in \mathbb{N}$.

Additive machines (in the sequel called simply "machines") are BSS machines whose computation nodes perform only additions and subtractions (see [4, 24] for details). To a machine M we naturally associate an input-output map $\varphi_M : \mathbb{R}^\infty \to$

$\mathbb{R}^\infty$. We shall say that a function $f : \mathbb{R}^\infty \to \mathbb{R}^k$, $k \leq \infty$, is *computable* when there is a machine M such that $f = \varphi_M$. Also, a set $A \subseteq \mathbb{R}^\infty$ is *decided* by a machine M if its characteristic function $\chi_A : \mathbb{R}^\infty \to \{0,1\}$ coincides with φ_M. So, for decision problems, we consider machines whose output space is $\{0,1\} \subset \mathbb{R}$.

We can now introduce some central complexity classes.

DEFINITION 2.1. *A machine M over $\mathbb{R}$ is said* to work in polynomial time *when there is a constant $c \in \mathbb{N}$ such that for every input $x \in \mathbb{R}^\infty$, M reaches its output node after at most* $\mathrm{size}(x)^c$ *steps. The class* $\mathrm{P}_{\mathrm{add}}$ *is then defined as the sets of all subsets of $\mathbb{R}^\infty$ that can be decided by a machine working in polynomial time. The class* $\mathrm{FP}_{\mathrm{add}}$ *is the class of functions computed by machines working in polynomial time.*

DEFINITION 2.2. *A set A belongs to* $\mathrm{NP}_{\mathrm{add}}$ *if there is a machine M satisfying the following condition: for all x, $x \in A$ iff there is $y \in \mathbb{R}^\infty$ such that M accepts the input (x, y) within time polynomial in* $\mathrm{size}(x)$. *In this case, the element y is said to be a* witness *for x. If we require the witness y to belong to $\{0,1\}^\infty$ we say that $A \in \mathrm{DNP}_{\mathrm{add}}$ (the D standing for* digital*). Abusing language we will call the machine M above an* $\mathrm{NP}_{\mathrm{add}}$-machine *(resp., a* $\mathrm{DNP}_{\mathrm{add}}$*-machine).*

Remark 2.3.

(i) In this model the element y can be seen as the sequence of guesses used in the Turing machine model (but note that in the case of $\mathrm{NP}_{\mathrm{add}}$ these guesses are not necessarily binary). However, we note that in this definition no nondeterministic machine is introduced as a computational model, and nondeterminism appears here as a new acceptance definition for the deterministic machine. Also, we note that w.l.o.g., the length of y can be bounded by the running time of M (which is of the form $p(\mathrm{size}(x))$ for a polynomial p).

(ii) The definitions of $\mathrm{NP}_{\mathrm{add}}$ and $\mathrm{DNP}_{\mathrm{add}}$ extend in a straightforward manner to all levels of the polynomial hierarchies $\mathrm{PH}_{\mathrm{add}}$ and $\mathrm{DPH}_{\mathrm{add}}$, respectively (i.e., to the classes Σ^k_{add} and Π^k_{add} for $k \geq 0$). For details see [4, 14].

DEFINITION 2.4. *We say that an additive machine* has no real constants *when the only machine constants appearing in its program are* 0 *and* 1. *Complexity classes for these machines are distinguished by the superscript* 0 *as in* $\mathrm{P}^0_{\mathrm{add}}$, $\mathrm{NP}^0_{\mathrm{add}}$.

Natural examples of sets in these classes exist. For instance, the real traveling salesman problem $\mathrm{TSP}_{\mathbb{R}}$ discussed in section 1.1 belongs to $\mathrm{DNP}_{\mathrm{add}}$ (actually to $\mathrm{DNP}^0_{\mathrm{add}}$). Problems which are known to be $\mathrm{NP}_{\mathrm{add}}$-complete for many-one reductions are scarce (for some known problems see [15]). In contrast, the following result by Fournier and Koiran [17] exhibits plenty of $\mathrm{NP}_{\mathrm{add}}$-complete problems with respect to Turing reductions. For a problem $S \subseteq \mathbb{R}^\infty$, we define its *integer part* to be $S_{\mathbb{Z}} = S \cap \mathbb{Z}^\infty$.

THEOREM 2.5 (see [17]). *Let $S \in \mathrm{NP}_{\mathrm{add}}$. If $S_{\mathbb{Z}}$ is* NP*-complete with respect to Turing reductions, then S is* $\mathrm{NP}_{\mathrm{add}}$*-complete with respect to Turing reductions.*

Of course, this theorem implies the $\mathrm{NP}_{\mathrm{add}}$-completeness for Turing reductions of a large number of decision problems, for instance for $\mathrm{TSP}_{\mathbb{R}}$ and $\mathrm{PM}_{\mathbb{R}}$ (for formal definitions of these problems, see section 6). Note that, in particular, every discrete NP-complete problem (e.g., SAT) is $\mathrm{NP}_{\mathrm{add}}$-Turing-complete.

A basic fact used in proving many results on additive machines is the existence of "small" rational points in polyhedra when the defining matrix has "small" integer entries. In what follows, for an integer $n \geq 1$, we denote the set $\{1, \ldots, n\}$ by $[n]$.

THEOREM 2.6 (see Theorem 3, Chapter 21 of [4]). *Let P be a nonempty polyhedron of $\mathbb{R}^n$ defined by a system*

$$A_1 y \leq b_1; \ A_2 y < b_2, \tag{2.1}$$

where $A_1 \in \mathbb{Z}^{N_1 \times n}, A_2 \in \mathbb{Z}^{N_2 \times n}, b_1 \in \mathbb{R}^{N_1}$, *and* $b_2 \in \mathbb{R}^{N_2}$. *The entries of* A_1 *and* A_2 *are integers of bit-size bounded by* L. *Then there is* $y \in P$ *with the following description:*

$$y_i = \sum_{j \in I_1} u_{ij} b_{1j} + \sum_{j \in I_2} v_{ij} b_{2j} + w_i, \qquad i = 1, \ldots, n,$$

where $I_1 \subseteq [N_1]$, $I_2 \subseteq [N_2]$, $|I_1| + |I_2| \leq n$, *and the coefficients* u_{ij}, v_{ij}, w_i *are rationals of bit-size at most* $(Ln)^c$ *for some constant* c.

Remark 2.7. The feasibility of a system (2.1) of linear inequalities for given integer matrices A_1, A_2 and real vectors b_1, b_2 can be decided in $\mathrm{P}_{\mathrm{add}}$. Moreover, a solution can be computed in $\mathrm{FP}_{\mathrm{add}}$, if it exists. This is just a rephrasing a of well-known and important result by Tardos [37]. We will not need this remark in the rest of the paper.

Recall from [4] or [10, Exercise 3.15] that a *linear decision tree* T is a regular binary tree, whose internal nodes are labeled by linear functions $\ell : \mathbb{R}^n \to \mathbb{R}$, and whose leaves are labeled with "accept" or "reject." Here, n is the dimension of the input space. At a given node, an input $x \in \mathbb{R}^n$ goes to the left child if $\ell(x) \geq 0$ and to the right child if $\ell(x) < 0$. Let $X \subseteq \mathbb{R}^n$ be the set accepted by the linear decision tree T. The set of inputs in $\mathbb{R}^n$, whose path in the computation tree T ends up with a specific leaf ν, shall be called the *leaf set* D_ν of ν. Note that the set D_ν can be described by a set of linear inequalities and is therefore convex. It is clear that the leaf sets corresponding to the accepting leaves form a partition of the set X. In particular, X is semilinear (cf. section 1.4).

We next use Theorem 2.6 to prove that $\mathrm{NP}_{\mathrm{add}} = \mathrm{DNP}_{\mathrm{add}}$. This is a well-known result [24], but the idea of the proof will be repeatedly used in this paper.

COROLLARY 2.8. $\mathrm{NP}_{\mathrm{add}} = \mathrm{DNP}_{\mathrm{add}}$.

Proof. Let $X \in \mathrm{NP}_{\mathrm{add}}$ and M be a machine deciding X as in Definition 2.2. By unwinding the computation of M on pairs $(x, y) \in \mathbb{R}^n \times \mathbb{R}^{p(n)}$ we obtain a linear decision tree T of depth polynomial in n. If z is a value tested for positivity at a branch node of this tree, then

$$z = \sum_{i=1}^{p(n)} a_i y_i + \sum_{i=1}^{n} b_i x_i + \sum_{i=1}^{k} c_i \alpha_i + d, \tag{2.2}$$

where $\alpha_1, \ldots, \alpha_k$ are the constants of M and the coefficients a_i, b_i, c_i, and d are integers of bit-size polynomial in n. Thus, for a given $x \in \mathbb{R}^n$, the leaf set D_ν of points y such that (x, y) reaches the accepting leaf ν in T is the set of solutions of a system of inequalities as in Theorem 2.6. We conclude that D_ν is nonempty iff D_ν contains a point y such that, for $i = 1, \ldots, p(n)$,

$$y_i = \sum_{j=1}^{n} b_{ij} x_j + \sum_{j=1}^{k} c_{ij} \alpha_j + d_i, \tag{2.3}$$

where the coefficients b_{ij}, c_{ij}, and d_i are rationals of bit-size polynomial in n (for a polynomial which does not depend on ν or x). Then, to decide whether $x \in X$, one can guess $b_{ij}, c_{ij}, d_i \in \mathbb{Q}$, compute y_i according to (2.3), and check whether M accepts $(x_1, \ldots, x_n, y_1, \ldots, y_{p(n)})$. Alternatively, one could also compute y in $\mathrm{FP}_{\mathrm{add}}$ according to Remark 2.7 and check whether M accepts $(x_1, \ldots, x_n, y_1, \ldots, y_{p(n)})$. □

Over the real numbers, space is not as meaningful a resource as it is in the discrete setting (cf. [31]). The role of space, however, is satisfactorily played by parallel time (cf. [4, 14]). To introduce parallel time, we briefly recall the model of additive circuits, a restriction of the more general model of arithmetic circuits introduced in [19].

DEFINITION 2.9. *An* additive circuit $\mathcal{C}$ *over* $\mathbb{R}$ *is an acyclic directed graph where each node has indegree* 0, 1, 2, *or* 3. *Nodes with indegree* 0 *are either labeled as* input nodes *or with elements of* $\mathbb{R}$ *(we shall call these* constant nodes*). Nodes with indegree* 2 *are labeled with one of* $\{+, -\}$. *They are called* arithmetic nodes. *Nodes with indegree* 1 *are* output nodes. *Nodes with indegree* 3 *are* selection nodes. *All output nodes have outdegree* 0. *Otherwise, there is no upper bound on the outdegree of the other nodes.*

For an additive circuit $\mathcal{C}$, *the* size *of* $\mathcal{C}$ *is the number of nodes in* $\mathcal{C}$. *The* depth *of* $\mathcal{C}$ *is the length of the longest path from some input node to some output node.*

The semantics of a selection node is as follows. With input (x, y, z) the node returns y if $x \geq 0$ and z otherwise. The semantics of all other nodes is obvious. If $\mathcal{C}$ is an additive circuit with n input nodes and m output nodes, we may talk about the function $\varphi_{\mathcal{C}} : \mathbb{R}^n \to \mathbb{R}^m$ *computed by the circuit.* We remark that the computation of an additive circuit can always be unwound to a linear decision tree.

Let $f : \mathbb{R}^\infty \to \mathbb{R}^\infty$. The family of additive circuits $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ *computes* f if for all $n \geq 1$, $\varphi_{\mathcal{C}_n}$ is the restriction of f to $\mathbb{R}^n$.

The other ingredient we need to define parallel complexity classes is a notion of uniformity.

Note that nodes of additive circuits can be described by five real numbers in the following way. If the nodes of the circuit are $g_1, \ldots, g_N$, then node g_j is described by the tuple $(j, t, i_\ell, i_r, i_m) \in \mathbb{R}^5$, where t represents the type of g_j according to the following (arbitrary) dictionary:

g_j	input	constant	+	−	selection	output
t	1	2	3	4	5	6

.

For nodes of indegree 2, i_ℓ and i_r denote the numbers of the nodes which provide left and right input to g_j, respectively. If g_j is a constant node, then i_ℓ equals its constant, and if g_j is an output node, then i_ℓ numbers the node which provides the input to g_j. Finally, if g_j is a selection node, then i_ℓ, i_r, and i_m number its left, right, and middle inputs. All components not mentioned above are set to 0. Thus, the whole circuit can be described by a point in $\mathbb{R}^{5N}$.

DEFINITION 2.10. *A family of circuits* $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ *is said to be* uniform *if there exists an additive machine* M *that, on input* (n, i), *outputs the description of the* i*th node of* $\mathcal{C}_n$. *If* M *works in time* $n^{\mathcal{O}(1)}$, *we shall say that the family is* $\mathrm{P}_{\mathrm{add}}$-uniform.

We denote by $\mathrm{PAR}_{\mathrm{add}}$ the class of decision problems whose characteristic function can be computed in parallel polynomial time, i.e., by a $\mathrm{P}_{\mathrm{add}}$-uniform family of circuits such that $\mathrm{depth}(\mathcal{C}_n) = n^{\mathcal{O}(1)}$. Also, $\mathrm{FPAR}_{\mathrm{add}}$ denotes the class of functions f which can be computed with such resources and for which there is a polynomial p such that $\mathrm{size}(f(x)) = p(\mathrm{size}(x))$ for all $x \in \mathbb{R}^\infty$.

Remark 2.11.

(i) Corollary 2.8 extends to all the polynomial hierarchy. That is, the power of real quantification is the same as that of digital quantification as long as the number of quantifier alternations is bounded. Surprisingly, if the number of quantifier alternations is not bounded, then the power of digital quantification is exactly $\mathrm{PAR}_{\mathrm{add}}$ and that of real quantification is at least additive exponential time, thus showing that the latter is more powerful than the former. For details see [4, 14].

(ii) In classical complexity theory, NP is a class of decision problems. Yet, if $S \in \mathsf{NP}$ and $x \in S$, a witness y for x can be computed in $\mathsf{FP}^{\mathsf{NP}}$ and thus in FPSPACE by computing its components one by one with an NP-routine. Looking at the proof of Corollary 2.8, we see that one can do the same with $\mathrm{NP}_{\mathrm{add}}$ and $\mathrm{FPAR}_{\mathrm{add}}$.

3. Counting classes. We now want to define counting classes, following the lines used in discrete complexity theory to define #P. This is the class of functions $f\colon \{0,1\}^\infty \to \mathbb{N}$ for which there exists an NP-machine M and a polynomial p such that, for all $n \in \mathbb{N}$, $x \in \{0,1\}^n$, $f(x) = |\{y \in \{0,1\}^{p(n)} \mid M \text{ accepts } (x,y)\}|$. That is, $f(x)$ is the number of witnesses for x. A first remark is that over the reals one can define two such complexity classes by counting the witnesses in an $\mathrm{NP}_{\mathrm{add}}$-machine or in a $\mathrm{DNP}_{\mathrm{add}}$-machine, respectively.

DEFINITION 3.1.

1. *We say that a function $f : \mathbb{R}^\infty \to \mathbb{N} \cup \{\infty\}$ belongs to the class $\#\mathrm{P}_{\mathrm{add}}$ if there exists a $\mathrm{NP}_{\mathrm{add}}$-machine M and a polynomial p such that, for all $n \in \mathbb{N}$, $x \in \mathbb{R}^n$,*

$$f(x) = |\{y \in \mathbb{R}^{p(n)} \mid M \text{ accepts } (x,y)\}|.$$

2. *We say that a function $f\colon \mathbb{R}^\infty \to \mathbb{N}$ belongs to the class $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$ if there exists a $\mathrm{DNP}_{\mathrm{add}}$-machine M and a polynomial p such that, for all $n \in \mathbb{N}$, $x \in \mathbb{R}^n$,*

$$f(x) = |\{y \in \{0,1\}^{p(n)} \mid M \text{ accepts } (x,y)\}|.$$

Remark 3.2.

(i) An unrestricted version of the class $\#\mathrm{P}_{\mathrm{add}}$ defined for machines over $\mathbb{R}$ which can multiply and divide was defined by Meer in [27].

(ii) Note that it is not clear that $\mathrm{NP}_{\mathrm{add}} = \mathrm{DNP}_{\mathrm{add}}$ implies $\#\mathrm{P}_{\mathrm{add}} = \mathrm{D}\#\mathrm{P}_{\mathrm{add}}$, since now we are counting witnesses instead of deciding their existence.

(iii) If f belongs to $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$, then the bit-size of $f(x)$ is bounded by a polynomial in the size of x. The same holds for $f \in \#\mathrm{P}_{\mathrm{add}}$ for those $x \in \mathbb{R}^n$ for which $f(x)$ is finite.

The next proposition locates the power of counting complexity classes within the landscape of known complexity classes. For interpreting the second inclusion, one should represent the value ∞ by some number in $\mathbb{R} \setminus \mathbb{N}$.

PROPOSITION 3.3. *We have the following inclusions of complexity classes over $\mathbb{R}$:*

$$\mathrm{D}\#\mathrm{P}_{\mathrm{add}} \subseteq \#\mathrm{P}_{\mathrm{add}} \subseteq \mathrm{FPAR}_{\mathrm{add}}.$$

To prove Proposition 3.3 we will use the following result. Let $\mathrm{CINF}_{\mathrm{add}}$ be the problem to decide whether the solution set described by an additive circuit has infinitely many points.

LEMMA 3.4. $\mathrm{CINF}_{\mathrm{add}}$ *is* $\mathrm{NP}_{\mathrm{add}}$*-complete.*

Proof. Recall from section 1.4 that the circuit satisfiability problem $\mathrm{CSAT}_{\mathrm{add}}$ is $\mathrm{NP}_{\mathrm{add}}$-complete. Adding a dummy variable to an additive circuit gives a (trivial) reduction from $\mathrm{CSAT}_{\mathrm{add}}$ to $\mathrm{CINF}_{\mathrm{add}}$, which shows the $\mathrm{NP}_{\mathrm{add}}$-hardness of $\mathrm{CINF}_{\mathrm{add}}$.

For the membership in $\mathrm{NP}_{\mathrm{add}}$, note that leaf sets are convex. So they are infinite iff they contain at least two points. Therefore, the following algorithm shows that membership of $\mathrm{CINF}_{\mathrm{add}}$ is in $\mathrm{NP}_{\mathrm{add}}$. On input $\mathcal{C}$ guess a leaf ν and guess $y^1, y^2 \in \mathbb{R}^n$. Then check whether $y^1 \neq y^2$ and whether y^1, y^2 reach the leaf ν. If yes, then accept; otherwise, reject. □

Proof of Proposition 3.3. The first inclusion is clear. For the second, consider the algorithm that, in parallel, checks for each accepting leaf ν whether there is any point in D_ν and, if yes, whether D_ν has infinitely many points. These verifications can be done in $\mathrm{NP}_{\mathrm{add}}$. If $|D_\nu| = \infty$ for some ν, then return ∞; else return the number of ν's such that $D_\nu \neq \emptyset$. This procedure clearly is in $\mathrm{FPAR}_{\mathrm{add}}^{\mathrm{NP}_{\mathrm{add}}} = \mathrm{FPAR}_{\mathrm{add}}$. ∎

We will see in Theorem 4.7 below that the difference in power of $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$ and $\#\mathrm{P}_{\mathrm{add}}$ is negligible.

We now focus on complete problems. To do so, we define the appropriate notions of reduction.

DEFINITION 3.5. *Let $f, g : \mathbb{R}^\infty \to \mathbb{N} \cup \{\infty\}$ and $\mathcal{C}$ be any of $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$ or $\#\mathrm{P}_{\mathrm{add}}$.*

1. *We say that $\varphi : \mathbb{R}^\infty \to \mathbb{R}^\infty$ is a* parsimonious reduction *from f to g if φ can be computed in polynomial time and, for all $x \in \mathbb{R}^\infty$, $f(x) = g(\varphi(x))$.*

2. *We say that f* Turing reduces *to g if there exists an oracle machine which, with oracle g computes f in polynomial time.*

3. *We say that a function g is $\mathcal{C}$-*hard *if, for every $f \in \mathcal{C}$, there is a parsimonious reduction from f to g. We say that g is $\mathcal{C}$-*complete *if, in addition, $g \in \mathcal{C}$.*

4. *The notions of hardness and completeness with respect to Turing reductions are defined similarly.*

Let $\#\mathrm{CSAT}_{\mathrm{add}}$ denote the problem of counting the number of points of a semilinear set given by an additive circuit. (Note that this requires computing a function with values in $\mathbb{N} \cup \{\infty\}$.)

THEOREM 3.6. *The counting problem $\#\mathrm{CSAT}_{\mathrm{add}}$ is $\#\mathrm{P}_{\mathrm{add}}$-complete.*

Proof. One just checks that the usual many-one reduction from the nondeterministic machine acceptance to additive circuit satisfiability is parsimonious. ∎

We close this section by recalling a principle introduced by Toda [38] and Toda and Watanabe [39], which allows us to assign to any complexity class $\mathcal{C}$ of decision problems a corresponding counting complexity class $\# \cdot \mathcal{C}$.

DEFINITION 3.7. *Given a set $A \in \{0,1\}^\infty$ and a polynomial p, we define the function $\#_A^p : \{0,1\}^\infty \to \mathbb{N}$ which associates to $x \in \{0,1\}^n$ the number*

$$\#_A^p(x) = |\{y \in \{0,1\}^{p(n)} \mid (x,y) \in A\}|.$$

If $\mathcal{C} \subseteq 2^{\{0,1\}^\infty}$ is a complexity class of decision problems, then we define

$$\# \cdot \mathcal{C} = \{\#_A^p \mid A \in \mathcal{C} \text{ and } p \text{ a polynomial}\}.$$

Similarly, one assigns $\# \cdot \mathcal{C}$ and $\mathrm{D}\# \cdot \mathcal{C}$ to a complexity class $\mathcal{C}$ over $\mathbb{R}$.

Note that $\# \cdot \mathsf{P} = \#\mathsf{P}$, $\# \cdot \mathrm{P}_{\mathrm{add}} = \#\mathrm{P}_{\mathrm{add}}$, and $\mathrm{D}\# \cdot \mathrm{P}_{\mathrm{add}} = \mathrm{D}\#\mathrm{P}_{\mathrm{add}}$.

We will use the following important result due to Toda and Watanabe several times.

THEOREM 3.8 (see [39]). *We have $\# \cdot \mathsf{PH} \subseteq \mathsf{FP}^{\#\mathsf{P}}$.*

4. Relationships between the real additive and the discrete setting. The work of Koiran [24], Cucker and Koiran [14], and Fournier and Koiran [16, 17] establishes close relationships between the real additive and the discrete model of computation. Building on these techniques, we show in section 4.1 that similar relationships hold for the counting classes. Then we use this in section 4.2 to derive transfer theorems for counting classes between the additive real and the discrete setting.

4.1. The power of discrete oracles. The main result of this section, Theorem 4.1 stated below, says that for several classical complexity classes $\mathcal{C}$ the corresponding additive complexity classes $\mathcal{C}_{\mathrm{add}}$ are contained in, or even equal to, $\mathrm{P}^{\mathcal{C}}_{\mathrm{add}}$. That is, all problems in $\mathcal{C}_{\mathrm{add}}$ can be solved by an additive machine working in polynomial time and having access to an oracle in $\mathcal{C}$.

This result was stated and proved for the class $\mathrm{NP}_{\mathrm{add}}$ in [17]. Moreover, in [17, Remark 2] it was already mentioned that the result for $\mathrm{NP}_{\mathrm{add}}$ can be extended to the classes of the polynomial hierarchy and to $\mathrm{PAR}_{\mathrm{add}}$. So what is new in Theorem 4.1 is the extension to the counting classes and to the functional class $\mathrm{FPAR}_{\mathrm{add}}$.

THEOREM 4.1. *The following statements hold* ($k \geq 0$):

1. $\Sigma^k_{\mathrm{add}} \subseteq \mathrm{P}^{\Sigma^k}_{\mathrm{add}}$, $\Pi^k_{\mathrm{add}} \subseteq \mathrm{P}^{\Pi^k}_{\mathrm{add}}$, $\mathrm{PH}_{\mathrm{add}} = \mathrm{P}^{\mathsf{PH}}_{\mathrm{add}}$.
2. $\mathrm{D}\#\cdot\Sigma^k_{\mathrm{add}} \subseteq \mathrm{FP}^{\#\cdot\Sigma^k}_{\mathrm{add}}$, $\mathrm{D}\#\cdot\Pi^k_{\mathrm{add}} \subseteq \mathrm{FP}^{\#\cdot\Pi^k}_{\mathrm{add}}$, $\mathrm{D}\#\cdot\mathrm{PH}_{\mathrm{add}} \subseteq \mathrm{FP}^{\#\cdot\mathsf{PH}}_{\mathrm{add}}$.
3. $\mathrm{PAR}_{\mathrm{add}} = \mathrm{P}^{\mathsf{PSPACE}}_{\mathrm{add}}$.
4. $\mathrm{FPAR}_{\mathrm{add}} = \mathrm{FP}^{\mathrm{PAR}_{\mathrm{add}}}_{\mathrm{add}} = \mathrm{FP}^{\mathsf{PSPACE}}_{\mathrm{add}}$.

Observe that part 2 of this theorem implies that $\mathrm{D}\#\mathrm{P}_{\mathrm{add}} \subseteq \mathrm{FP}^{\#\mathsf{P}}_{\mathrm{add}}$.

As in Fournier and Koiran [16, 17], the proof of Theorem 4.1 relies on Meyer auf der Heide's (nonuniform) construction of small depth linear decision trees for point location in arrangements of hyperplanes [29, 30]. Before giving the proof, we need to develop some lemmas.

First, let us recall some terminology regarding arrangements of hyperplanes. For $s, n \in \mathbb{N}$ we define $\mathcal{H}_{s,n}$ to be the set of linear polynomials $a_0 + \sum_{i=1}^n a_i X_i$ with integer coefficients a_i such that $\sum_{i=0}^n |a_i| \leq 2^s$. We denote by $\mathcal{F}_{s,n}$ the set of all nonempty sets

$$F = \bigcap_{f \in \mathcal{H}_{s,n}} \{x \in \mathbb{R}^n \mid f(x) = \sigma(f)\}$$

corresponding to some sign function $\sigma\colon \mathcal{H}_{s,n} \to \{-1, 0, 1\}$. The space $\mathbb{R}^n$ is the disjoint union of all $F \in \mathcal{F}_{s,n}$. We will call this the *universal cell decomposition* for the parameters s, n, and we call the sets $F \in \mathcal{F}_{s,n}$ the corresponding *faces* or *cells*.

By Theorem 2.6, each face $F \in \mathcal{F}_{s,n}$ contains a rational point of bit-size at most $(sn)^c$ for some fixed constant $c > 0$ (even though a face F may be described by a number of constraints exponential in n, s). Therefore, $\log |\mathcal{F}_{s,n}| \leq (sn)^c$.

In what follows, we will encode a face $F \in \mathcal{F}_{s,n}$ by a triple $(s, n, x) \in \mathbb{N}^2 \times \mathbb{Q}^n$ such that $x \in F$ and the bit-size of x is at most $(sn)^c$. This way, we can describe all faces in $\mathcal{F}_{s,n}$, but, of course, the description is not unique. Abusing notation, we will shortly express this by saying that the face F is represented by a "small rational point" x.

For a fixed polynomial t we define $\mathcal{H}_t$ as the union of the $\mathcal{H}_{t(n),n}$ over all $n \in \mathbb{N}$ and $\mathcal{F}_t$ as the union of the $\mathcal{F}_{t(n),n}$ over all $n \in \mathbb{N}$.

LEMMA 4.2. *Let M be an additive machine without real constants taking inputs in $\mathbb{R}^\infty \times \{0,1\}^\infty$ such that its running time is bounded by a polynomial t in the size of its first argument. The discrete relation*

$$R := \{(F, y) \in \mathcal{F}_t \times \{0,1\}^\infty \mid \forall x \in F \ \big(M \text{ accepts } (x, y)\big)\}$$

can be checked in P*, that is, in classical polynomial time.*

Proof. Running M on an input $(x, y) \in \mathbb{R}^n \times \{0,1\}^m$ takes at most $t(n)$ steps by assumption. On such an input, the machine M branches according to the signs of

expressions $a_0 + \sum_{i=1}^n a_i x_i + \sum_{j=1}^m b_j y_j$, where $\sum_{i=0}^n |a_i| + \sum_{j=1}^m |b_j| \le 2^{t(n)}$. Hence the hyperplane defined by $(a_0 + \sum_{j=1}^m b_j y_j) + \sum_{i=1}^n a_i X_i$ belongs to $\mathcal{H}_{t(n),n}$ for any $y \in \{0,1\}^m$. It follows that if x and x' belong to the same face F of $\mathcal{F}_{t(n),n}$, then for all y, (x,y) and (x',y) follow the same path in the decision tree induced by M. Therefore, if M accepts (x,y) for some $x \in F$, then it must accept (x,y) for all $x \in F$.

Therefore, checking that $(F,y) \in R$ can be done as follows. Let F be represented by the small rational point $x \in F$. Simulate the computation of the real machine M on input (x,y) by a Turing machine. Since M has no real constants and works in polynomial time, this simulation takes polynomial time (see [24, 4]). □

The following is an immediate consequence of Lemma 4.2 and the definition of Σ^k.

COROLLARY 4.3. *Let M be an additive machine as in Lemma 4.2, $k \in \mathbb{N}$, and $q_1, \dots, q_k$ be polynomials. Consider the discrete relation*

$$R := \{(F,y) \in \mathcal{F}_t \times \{0,1\}^\infty \mid \forall x \in F\ \exists z_1 \forall z_2 \dots Q z_k \\ \big(M \text{ accepts } (x, y, z_1, \dots, z_k)\big)\},$$

where quantifiers alternate (Q is either existential or universal depending on the parity of k) and the quantification is over $z_i \in \{0,1\}^{q_i(\mathrm{size}(x))}$. Then R can be checked in (classical) Σ^k.

Consider the following problem $\mathrm{FEVAL}_{\mathrm{add}}$: given a quantifier-free formula ψ of the first-order theory of $(\mathbb{R}, +, -, \le)$ with k free variables and a point $x \in \mathbb{R}^k$, decide whether $\psi(x)$ holds. Note that the formula ψ can be encoded as an element of $\mathbb{R}^\infty$ in a straightforward way. In the following, we will identify ψ with its encoding. It is well known that $\mathrm{FEVAL}_{\mathrm{add}} \in \mathrm{P}^0_{\mathrm{add}}$.

The next result is proved similarly as Lemma 4.2.

LEMMA 4.4. *Let M be a machine solving $\mathrm{FEVAL}_{\mathrm{add}}$ in time bounded by a polynomial t in $\mathrm{size}(\psi)$. Then the following set belongs to* PSPACE:

$$L := \{F \in \mathcal{F}_t \mid \forall \psi \in F\ Q_1 z_1 Q_2 z_2 \dots Q_n z_n\ \big(M \text{ accepts } (\psi, z_1, \dots, z_n)\big)\}.$$

Here $Q_i \in \{\forall, \exists\}$, $z_i \in \{0,1\}$. Moreover, n denotes the size of ψ. Hence the number of free variables of ψ is at most n, and the behavior of M on $(\psi, z_1, \dots, z_n)$ is well defined.

Given a polynomial t, the *point location problem* for t is the problem of computing, for a given input $x \in \mathbb{R}^\infty$, a small rational point of the uniquely determined face $F \in \mathcal{F}_{t(\mathrm{size}(x)),\mathrm{size}(x)}$ in which x lies. The following crucial statement is proved by Fournier and Koiran [17, Theorem 2].

PROPOSITION 4.5. *For any polynomial t, the point location problem can be solved in $(\mathrm{FP}^0_{\mathrm{add}})^{\mathsf{NP}}$. That is, a small rational point of the face $F \in \mathcal{F}_{t(n),n}$ containing the input point can be computed in polynomial time by an additive machine using a classical oracle in* NP.

We remark that in [17] a face F is represented by a system $\mathcal{S}$ of $n^{\mathcal{O}(1)}$ linear inequalities with integer coefficients of bit-size $n^{\mathcal{O}(1)}$ such that the polyhedral set defined by the system $\mathcal{S}$ is nonempty and is contained in the face F. However, note that since linear programming (discrete setting) is in polyomial time [21, 22], we can always compute from the system $\mathcal{S}$ a small rational point of F in polynomial time.

Proof of Theorem 4.1. 1. Assume that $A \in \Sigma^k_{\mathrm{add}}$. Then (cf. Remark 2.11(i)) there exist polynomials $q_1, \dots, q_k$ and $B \in \mathrm{P}_{\mathrm{add}}$ such that for all $x \in \mathbb{R}^\infty$

$$x \in A \iff \exists z_1 \forall z_2 \dots Q z_k\ (x,z) \in B,$$

where quantifiers alternate (Q is either existential or universal depending on the parity of k) and the quantification is over $z_i \in \{0,1\}^{q_i(\mathrm{size}(x))}$.

Let M_B be an additive machine deciding B in time bounded by some polynomial t and $\alpha_1, \ldots, \alpha_\ell$ be the constants occurring in the program of M_B other than 0 or 1. Denote $\alpha = (\alpha_1, \ldots, \alpha_\ell)$, and let

$$C = \{(x,z,v) \in \mathbb{R}^\infty \times \{0,1\}^\infty \times \mathbb{R}^\ell \mid M_B \text{ accepts } (x,z) \text{ when replacing } \alpha \text{ by } v \text{ in its program}\}.$$

Clearly, $C \in \mathrm{P}^0_{\mathrm{add}}$ and, for all $(x,z) \in \mathbb{R}^\infty \times \{0,1\}^\infty$, $(x,z) \in B \iff (x,z,\alpha) \in C$. In order to prove that $A \in \mathrm{P}^{\Sigma^k}_{\mathrm{add}}$, it is sufficient to show that the set

$$A' = \{(x,v) \in \mathbb{R}^\infty \times \mathbb{R}^\ell \mid \exists z_1 \forall z_2 \ldots Q z_k \ (x,z,v) \in C\}$$

belongs to the class $(\mathrm{P}^0_{\mathrm{add}})^{\Sigma^k}$. This reasoning shows that we may assume w.l.o.g. that M_B has no real constants, i.e., that $B \in \mathrm{P}^0_{\mathrm{add}}$.

Since $B \in \mathrm{P}^0_{\mathrm{add}}$, we may use Corollary 4.3 (used without the y in the input) to deduce that the discrete language

$$R = \{F \in \mathcal{F}_t \mid \forall x \in F \ \exists z_1 \forall z_2 \ldots Q z_k \ \big(M_B \text{ accepts } (x,z)\big)\}$$

lies in the class Σ^k. (Recall that t bounds the running time of M_B.) Consider now the following algorithm. On input $x \in \mathbb{R}^n$ locate x in a face F of $\mathcal{F}_{t(n),n}$ (due to Proposition 4.5, this can be done in $(\mathrm{FP}^0_{\mathrm{add}})^{\mathsf{NP}}$). Then decide whether $F \in R$ by an oracle call to Σ^k. This algorithm works in $\mathrm{P}^{\Sigma^k}_{\mathrm{add}}$ and decides A.

The above reasoning shows that $A \in \mathrm{P}^{\Sigma^k}_{\mathrm{add}}$. This immediately implies the inclusions for Π^k and PH. And, since $\mathrm{PH}_{\mathrm{add}}$ is closed under Turing reductions, we even get equality in this case.

2. Let $\varphi\colon \mathbb{R}^\infty \to \mathbb{N}$ be a counting problem in $\mathrm{D}\#\cdot\Sigma^k_{\mathrm{add}}$. Then there exist $B \in \mathrm{P}_{\mathrm{add}}$ and polynomials $p, q_1, \ldots, q_k$ such that for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}^n$

$$\varphi(x) = |\{y \in \{0,1\}^{p(n)} \mid \exists z_1 \forall z_2 \ldots Q z_k \ (x,y,z_1,\ldots,z_k) \in B\}|,$$

where quantifiers alternate and the quantification is over $z_i \in \{0,1\}^{q_i(\mathrm{size}(x))}$.

Let M_B be some additive machine deciding B in time t for some polynomial t. As in the proof of part 1, we can assume that M_B has no real constants. By Corollary 4.3, the map $\psi\colon \mathcal{F}_t \to \mathbb{N}$ defined for $F \in \mathcal{F}_{t(n),n}$ by

$$\psi(F) := |\{y \in \{0,1\}^{p(n)} \mid \forall x \in F \ \exists z_1 \forall z_2 \ldots Q z_k \ (x,y,z_1,\ldots,z_k) \in B\}|$$

lies in the discrete counting class $\# \cdot \Sigma^k$. Note that $\varphi(x) = \psi(F)$ for $F \in \mathcal{F}_{t(n),n}$, $x \in F$.

Consider now the following algorithm. On input $x \in \mathbb{R}^n$, locate x in a face F of $\mathcal{F}_{t(n),n}$. Then compute $\psi(F)$ by an oracle call to $\# \cdot \Sigma^k$ and return $\psi(F)$. This algorithm works in $\mathrm{FP}^{\#\cdot\Sigma^k}_{\mathrm{add}}$ and computes $\varphi(x)$.

The above proves the inclusion $\mathrm{D}\# \cdot \Sigma^k_{\mathrm{add}} \subseteq \mathrm{FP}^{\#\cdot\Sigma^k}_{\mathrm{add}}$. The inclusion $\mathrm{D}\# \cdot \Pi^k_{\mathrm{add}} \subseteq \mathrm{FP}^{\#\cdot\Pi^k}_{\mathrm{add}}$ is proved similarly, and it follows that $\mathrm{D}\# \cdot \mathrm{PH}_{\mathrm{add}} \subseteq \mathrm{FP}^{\#\cdot\mathsf{PH}}_{\mathrm{add}}$.

3. The inclusion $\mathrm{P}^{\mathsf{PSPACE}}_{\mathrm{add}} \subseteq \mathrm{PAR}_{\mathrm{add}}$ is clear, since $\mathrm{PAR}_{\mathrm{add}}$ is closed under Turing reductions. Consider the subset DTRAO of the theory of the reals with addition and order which consists of the sentences all of whose variables z satisfy a constraint of

the form $z = 0 \vee z = 1$. In [14] it is proven that DTRAO is $\mathrm{PAR}_{\mathrm{add}}$-complete. It is therefore sufficient to show that $\mathrm{DTRAO} \in \mathrm{P}_{\mathrm{add}}^{\mathsf{PSPACE}}$.

Consider now the following algorithm. On input a digitally quantified first-order sentence $\varphi = Q_1 z_1 Q_2 z_2 \dots Q_k z_k \ \psi(z_1, \dots, z_k)$ of the theory of $(\mathbb{R}, +, -, \leq)$, where ψ is quantifier-free of size n, we locate ψ in a face $F \in \mathcal{F}_{t(n),n}$, where n is the size of ψ. By construction, φ is true iff F belongs to the set L in Lemma 4.4. We then decide this membership by an oracle call to PSPACE. This algorithm decides $\mathrm{DTRAO} \in \mathrm{P}_{\mathrm{add}}^{\mathsf{PSPACE}}$.

4. The equality $\mathrm{FP}_{\mathrm{add}}^{\mathrm{PAR}_{\mathrm{add}}} = \mathrm{FP}_{\mathrm{add}}^{\mathsf{PSPACE}}$ follows from the third statement of Theorem 4.1. To prove the first equality, let $f \in \mathrm{FPAR}_{\mathrm{add}}$, $x \in \mathbb{R}^n$, and $y = f(x) \in \mathbb{R}^{p(n)}$. Let $\alpha_1, \dots, \alpha_k$ be the constants occurring in the additive machine that generates the circuits computing f as described in Definition 2.10. For $\ell \leq p(n)$ we have

$$y_\ell = \sum_{i=1}^{n} u_i^{(\ell)} x_i + \sum_{j=1}^{k} v_j^{(\ell)} \alpha_j + b^{(\ell)}, \tag{4.1}$$

where $u_i^{(\ell)}, v_j^{(\ell)}$, and $b^{(\ell)}$ are integers of bit-size at most $q(n)$ for a polynomial q. Let B_x be the relation defined by

$$B_x = \{(s, i, \ell, x) \in \mathbb{N}^3 \times \mathbb{R}^\infty \mid \text{ the } s\text{th bit of } u_i^{(\ell)} \text{ is } 1\}$$

and B_α, B_1 the analogous relations for $v_j^{(\ell)}$ and $b^{(\ell)}$. We claim that $B_x, B_\alpha, B_1 \in \mathrm{PAR}_{\mathrm{add}}$. In fact, the parallel algorithm deciding any of these relations simulates the behavior of $\mathcal{C}_n$ (the circuit computing the restriction of f to $\mathbb{R}^n$) on input x, keeping expressions in the form (4.1) instead of actually performing the arithmetic operations.

To compute $f(x)$ in $\mathrm{FP}_{\mathrm{add}}^{\mathrm{PAR}_{\mathrm{add}}}$, one uses the oracles B_x, B_α, and B_1 to obtain the binary expansions of $u_i^{(\ell)}$, $v_j^{(\ell)}$, and $b^{(\ell)}$ for all ℓ, i, j. Then compute y by (4.1). □

The following corollary is a real analogue of Toda and Watanabe's Theorem 3.8.

COROLLARY 4.6. *We have* $\mathrm{D}\#\cdot\mathrm{PH}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}}^{\#\mathsf{P}}$.

Proof. We conclude from Theorem 4.1(2) and Theorem 3.8 that

$$\mathrm{D}\#\cdot\mathrm{PH}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}}^{\#\cdot\mathsf{PH}} \subseteq \mathrm{FP}_{\mathrm{add}}^{\mathsf{FP}^{\#\mathsf{P}}} = \mathrm{FP}_{\mathrm{add}}^{\#\mathsf{P}},$$

which proves the claim. □

We use this to prove that the counting class $\#\mathrm{P}_{\mathrm{add}}$ is closely related to its digital variant $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$ in the sense that a $\#\mathrm{P}_{\mathrm{add}}$-oracle does not give more power to an additive polynomial time Turing machine than a $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$-oracle.

THEOREM 4.7. *We have* $\mathrm{FP}_{\mathrm{add}}^{\#\mathrm{P}_{\mathrm{add}}} = \mathrm{FP}_{\mathrm{add}}^{\mathrm{D}\#\mathrm{P}_{\mathrm{add}}} = \mathrm{FP}_{\mathrm{add}}^{\#\mathsf{P}}$.

Proof. Clearly, it is enough to show that $\#\mathrm{P}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}}^{\#\mathsf{P}}$. For this, it is sufficient to prove that

$$\#\mathrm{P}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}}^{\mathrm{D}\#\cdot\mathrm{NP}_{\mathrm{add}}}. \tag{4.2}$$

Indeed, by Corollary 4.6 we know that $\mathrm{D}\#\cdot\mathrm{NP}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}}^{\#\mathsf{P}}$.

In order to prove (4.2), let $\varphi \in \#\mathrm{P}_{\mathrm{add}}$. Then there exist a polynomial p and an additive machine M working in polynomial time such that, for all $n \in \mathbb{N}$, $x \in \mathbb{R}^n$,

$$\varphi(x) = |\{y \in \mathbb{R}^{p(n)} \mid M \text{ accepts } (x, y)\}|.$$

Using Lemma 3.4 we can find out in $\mathrm{NP}_{\mathrm{add}}$ whether $\varphi(x)$ is infinite on input $x \in \mathbb{R}^n$. Assume then that $\varphi(x)$ is finite. Since leaf sets are convex, they are either infinite or consist of just one point. In the case that $\varphi(x)$ is finite, we have

$$\varphi(x) = |\{\nu \in \{0,1\}^{t(n)} \mid \exists y \in \mathbb{R}^{p(n)} \ \text{ input } (x, y) \text{ reaches the accepting leaf } \nu\}|,$$

since y is uniquely determined. Define $B := \cup_{n\in\mathbb{N}} B_n$, where

$$B_n := \{(x,\nu) \in \mathbb{R}^n \times \{0,1\}^{t(n)} \mid \exists y \in \mathbb{R}^{p(n)} \text{ input } (x,y) \text{ reaches the accepting leaf } \nu\}.$$

Then we have $B \in \mathrm{NP}_{\mathrm{add}}$. This implies that $\varphi \in \mathrm{D}\# \cdot \mathrm{NP}_{\mathrm{add}}$. □

Remark 4.8.

(i) Statements analogous to Theorem 4.1 hold in the constant-free setting, e.g.,

$$\mathrm{NP}^0_{\mathrm{add}} \subseteq (\mathrm{P}^0_{\mathrm{add}})^{\mathsf{NP}}, \quad \mathrm{D}\#\mathrm{P}^0_{\mathrm{add}} \subseteq (\mathrm{FP}^0_{\mathrm{add}})^{\#\mathsf{P}}, \quad \mathrm{FPAR}^0_{\mathrm{add}} = (\mathrm{FP}^0_{\mathrm{add}})^{\mathsf{PSPACE}}.$$

(ii) Similarly to Theorem 4.7, we have $\mathrm{FP}^{\#\Sigma^k_{\mathrm{add}}}_{\mathrm{add}} = \mathrm{FP}^{\mathrm{D}\#\cdot\Sigma^k_{\mathrm{add}}}_{\mathrm{add}} = \mathrm{FP}^{\#\mathsf{P}}_{\mathrm{add}}$ for $k \geq 0$.

4.2. Boolean parts and transfer theorems. A problem that has attracted much attention in real complexity is the computation of Boolean parts [8, 12, 13, 14, 24, 25]. Roughly speaking, this amounts to characterizing, in terms of classical complexity classes, the power of resource bounded machines over $\mathbb{R}$ when their inputs are restricted to be binary. In this section, we will be interested in the Boolean parts of counting classes.

DEFINITION 4.9. *Let $\mathcal{C}$ be a counting class over $\mathbb{R}$. Its* Boolean part *is the classical complexity class* $\mathrm{BP}(\mathcal{C}) = \{f : \{0,1\}^\infty \to \mathbb{N} \mid f = g_{|\{0,1\}^\infty} \text{ for some } g \in \mathcal{C}\}$.

PROPOSITION 4.10. *We have* $\mathrm{BP}(\mathrm{D}\#\mathrm{P}_{\mathrm{add}}) = \#\mathsf{P}/\mathsf{poly}$.

Proof. The proof closely follows that of [4, Theorem 2, Chapter 22]. Consider a function f in $\#\mathsf{P}/\mathsf{poly}$. There is a polynomial q and an advice function h such that $h(n)$ belongs to $\{0,1\}^{q(n)}$ for all n. Furthermore, there are an NP-machine M and a polynomial p such that M accepts for exactly $f(x)$ witnesses in $\{0,1\}^{p(n)}$ on input $(x,h(n))$ for all $x \in \{0,1\}^n$. Let us code in a single number $\xi_h \in \mathbb{R}$ the sequence of advices $h(1), h(2), \ldots$. Then we can consider a $\mathrm{DNP}_{\mathrm{add}}$-machine which, for each input $x \in \{0,1\}^n$, first produces the digits of ξ_h and obtains $h(n)$ and then simulates M on $(x,h(n))$. This shows that $\#\mathsf{P}/\mathsf{poly} \subseteq \mathrm{BP}(\mathrm{D}\#\mathrm{P}_{\mathrm{add}})$.

Conversely, let us consider a function f in the Boolean part of $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$ defined by a $\mathrm{DNP}_{\mathrm{add}}$-machine M with time bound q. The computation of M on inputs of size n is described by a linear decision tree T of depth $q(n)$. Therefore, if $\alpha_1, \ldots, \alpha_k$ are the real constants of M, then, for each $x \in \{0,1\}^n$, the test performed by T at a node i has the form $g_i(x,\alpha) \geq 0$ with

$$g_i(x,\alpha) = \sum_{j=1}^{n} a_{ij}x_j + \sum_{j=1}^{k} b_{ij}\alpha_j + c_i \tag{4.3}$$

and where a_{ij}, b_{ij}, and c_i are integers of bit-size polynomial in n. For a given $x \in \{0,1\}^n$, according to the outcome of the test (4.3), the point $\alpha \in \mathbb{R}^k$ satisfies then an inequality of the form $g_{i,x}(\alpha) \geq 0$ or $g_{i,x}(\alpha) < 0$, where $g_{i,x} \in \mathbb{Z}[Y_1, \ldots, Y_k]$ is defined by $g_{i,x}(y) = g_i(x,y)$. Let Φ be the system of all these linear inequalities for i varying over all branching nodes of T and x varying over the 2^n possible points of $\{0,1\}^n$. The system Φ is satisfied by α. Then, according to Theorem 2.6, there is a point $\beta_n \in \mathbb{Q}^k$ all of whose coordinates have polynomial bit-size in n, which also satisfies Φ. Thus, if we replace $\alpha = (\alpha_1, \ldots, \alpha_k)$ by β_n in the tree T, the path followed by any $x \in \{0,1\}^n$ will not change, and x will be accepted or rejected as in T.

Let us now consider the function $h : \mathbb{N} \to \mathbb{Q}^k$ defined by $h(n) = \beta_n$. Since the bit-size of β_n is polynomial in n we may interpret h as an advice function. The classical machine which, with input $(x, h(\mathrm{size}(x)))$, simulates the behavior of M with the constants $\alpha_1, \ldots, \alpha_k$ replaced by $h(\mathrm{size}(x))$ shows that $f \in \#\mathsf{P}/\mathsf{poly}$. □

Combining Proposition 4.10 with the results of section 4.1, we obtain the following corollary.

COROLLARY 4.11. *We have the following transfer results:*

1. $\#\mathrm{P}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}}$ *iff* $\mathrm{D}\#\mathrm{P}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}}$ *iff* $\#\mathsf{P} \subseteq \mathsf{FP/poly}$.
2. $\mathrm{FPAR}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}}^{\#\mathrm{P}_{\mathrm{add}}}$ *iff* $\mathsf{PSPACE} \subseteq \mathsf{P}^{\#\mathsf{P}}\mathsf{/poly}$.
3. *Similar equivalences hold for additive machines without constants and uniform classical complexity classes, respectively.*

Proof. The first equivalence of part 1 follows from Theorem 4.7. In the second equivalence of part 1, the direction from left to right follows from

$$\#\mathsf{P} \subseteq \#\mathsf{P/poly} = \mathrm{BP}(\mathrm{D}\#\mathrm{P}_{\mathrm{add}}) \subseteq \mathrm{BP}(\mathrm{FP}_{\mathrm{add}}) \subseteq \mathsf{FP/poly}.$$

Here, the equality holds by Proposition 4.10, the last inclusion holds by Remark 4.12 below, and the one before the last is true by assumption. For the other direction, we use Theorem 4.1(2) to get $\mathrm{D}\#\mathrm{P}_{\mathrm{add}} \subseteq \mathrm{FP}_{\mathrm{add}}^{\#\mathsf{P}} \subseteq \mathrm{FP}_{\mathrm{add}}^{\mathsf{FP/poly}} = \mathrm{FP}_{\mathrm{add}}$, where the second inclusion follows by assumption. The other equivalences can be shown similarly. ∎

Remark 4.12. One can extend the definition of Boolean parts to classes of (not necessarily counting) functions over the reals and consider, for instance,

$$\mathrm{BP}(\mathrm{FP}_{\mathrm{add}}) = \{f : \{0,1\}^\infty \to \{0,1\}^\infty \mid f = g_{|\{0,1\}^\infty} \text{ for some } g \in \mathrm{FP}_{\mathrm{add}}\}.$$

One can then use the same arguments to show that

$$\mathrm{BP}(\mathrm{FP}_{\mathrm{add}}) = \mathsf{FP/poly},\ \mathrm{BP}(\mathrm{FP}_{\mathrm{add}}^{\#\mathrm{P}_{\mathrm{add}}}) = \mathsf{FP}^{\#\mathsf{P}}\mathsf{/poly},\ \mathrm{BP}(\mathrm{FPAR}_{\mathrm{add}}) = \mathsf{FPSPACE/poly}.$$

Also, for machines without constants we have the following, easier to prove, results:

$$\mathrm{BP}(\mathrm{FP}_{\mathrm{add}}^0) = \mathsf{FP},\ \mathrm{BP}((\mathrm{FP}_{\mathrm{add}}^0)^{\#\mathrm{P}_{\mathrm{add}}^0}) = \mathsf{FP}^{\#\mathsf{P}},\ \mathrm{BP}(\mathrm{FPAR}_{\mathrm{add}}^0) = \mathsf{FPSPACE}.$$

4.3. Complete counting problems. We obtain from Theorem 4.1 the following result, providing us with plenty of complete problems for the class $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$.

PROPOSITION 4.13. *Let $f\colon \mathbb{R}^\infty \to \mathbb{N}$ belong to $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$, and assume that the restriction of f to $\mathbb{Z}^\infty$ is $\#\mathsf{P}$-complete with respect to Turing reductions. Then f is $\#\mathrm{P}_{\mathrm{add}}$-complete and thus $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$-complete with respect to Turing reductions.*

Proof. This is an immediate consequence of Theorem 4.7. ∎

Proposition 4.13 yields plenty of Turing complete problems in $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$. We just mention two particularly interesting ones. Assume that we are given a graph G with real weights on the edges and some $w \in \mathbb{R}$. We define the weight of a subgraph as the sum of the weights of its edges.

1. *Counting traveling salesman.* Let $\#\mathrm{TSP}_{\mathbb{R}}$ be the problem to count the number of Hamilton cycles of weight at most w in the graph G.
2. *Counting weighted perfect matchings.* Let $\#\mathrm{PM}_{\mathbb{R}}$ be the problem to count the number of perfect matchings of weight at most w in the graph G (here we assume that G is bipartite).

Valiant [41, 42] proved the $\#\mathsf{P}$-completeness of the problem to count the number of Hamilton cycles of a given graph and of the problem to count the number of perfect matchings of a given bipartite graph. Together with Proposition 4.13, this immediately implies the following corollary.

COROLLARY 4.14. *The problems $\#\mathrm{TSP}_{\mathbb{R}}$ and $\#\mathrm{PM}_{\mathbb{R}}$ are $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$-complete with respect to Turing reductions.*

Note that the problem to count the number of perfect matchings of a given bipartite graph is equivalent to the famous problem to evaluate the *permanent* of a matrix with entries in $\{0,1\}$.

Remark 4.15.

1. Results for $\mathrm{PAR}_{\mathrm{add}}$ and $\mathrm{FPAR}_{\mathrm{add}}$ similar to Proposition 4.13 follow from Theorem 4.1 in the same manner.

2. There is an algebraic theory of NP-completeness due to Valiant [7, 40, 43], which captures the complexity to evaluate the generating functions of graph properties. For instance, the generating function of the property "perfect matching" is the permanent of a real matrix, which turns out to be complete in this theory. The functional problems studied in this theory take real values on real inputs and thus differ substantially from the counting problems studied in this paper. It would be interesting to clarify the relationship between these two approaches.

5. Complexity to compute topological invariants. In the computational problems studied in this section, it is always assumed that the input is an additive circuit $\mathcal{C}$ and X is the semilinear set accepted by $\mathcal{C}$.

In the following subsections, we characterize the complexity to compute several basic invariants of a semilinear set X. These invariants are the dimension (section 5.1), the number of connected components (section 5.2), the Euler characteristic (section 5.3.4), and the Betti numbers (section 5.3.5). In section 6 we show that corresponding completeness results for the Turing model hold as well.

5.1. Complexity of computing the dimension. For all $d \geq 0$, the problem $\mathrm{DIM}_{\mathrm{add}}(d)$ consists of deciding whether the set X given by an additive circuit $\mathcal{C}$ has dimension at least d. We define $\dim \emptyset := -1$ so that we can decide for nonemptiness using the dimension function.

THEOREM 5.1. *For all $d \geq 0$, the problem $\mathrm{DIM}_{\mathrm{add}}(d)$ is $\mathrm{NP}_{\mathrm{add}}$-complete.*

The main tool in proving Theorem 5.1 is the following proposition. It states that if a polyhedron P has dimension at least d, then there is a projection of P onto a d-dimensional coordinate subspace containing a scaled down d-dimensional standard simplex. This can be used to construct an $\mathrm{NP}_{\mathrm{add}}$-certificate.

To formally state this proposition we define $e_j = (0,\dots,0,1,0,\dots,0)^{\mathrm{T}} \in \mathbb{R}^n$, the column unit vector with 1 in the jth place. Also, recall that we denote by $[m]$ the set $\{1,\dots,m\}$, for any $m \in \mathbb{N}$.

PROPOSITION 5.2. *Let P be a polyhedron in $\mathbb{R}^n$ of dimension $d \geq 0$ defined by a system $A_1 x \leq b_1$; $A_2 x < b_2$, where $A_1 \in \mathbb{R}^{N_1 \times n}, A_2 \in \mathbb{R}^{N_2 \times n}, b_1 \in \mathbb{R}^{N_1}$, and $b_2 \in \mathbb{R}^{N_2}$. Then there exist points $x^{(0)},\dots,x^{(d)} \in P$, $\rho > 0$, and an injective map $\pi\colon [d] \to [n]$ such that for all $\ell, i \in [d]$ we have $x^{(\ell)}_{\pi(i)} - x^{(0)}_{\pi(i)} = \rho\delta_{\ell,i}$. Here δ is the Kronecker's delta.*

Proof. We first prove the result for the case $d = n$. Let $N = N_1 + N_2$, $A = \left[\begin{smallmatrix} A_1 \\ A_2 \end{smallmatrix}\right] \in \mathbb{R}^{N \times n}$, $b = \left[\begin{smallmatrix} b_1 \\ b_2 \end{smallmatrix}\right] \in \mathbb{R}^N$, and denote by $a_i^{\mathrm{T}} \in \mathbb{R}^n$ the ith row of $A = (a_{i,j})$. W.l.o.g. $A \neq 0$. Since $d = n$, the polyhedron

$$P^* := \{x \in \mathbb{R}^n \mid Ax < b\}$$

is nonempty. Let $x^* \in P^*$, and put $y^* := b - Ax^* > 0$, $\varepsilon := \min_{i \leq N} y_i^* = y_{i_0}^*$. Then $a_i^{\mathrm{T}} x^* \leq b_i - \varepsilon$ for all $i \leq N$. Define $\rho = \varepsilon/(2\max_{i,j} |a_{ij}|)$. For all i, j we have $\rho |a_i^{\mathrm{T}} e_j| = \rho |a_{ij}| \leq \varepsilon/2$. This implies that, for all i,

$$|a_i^{\mathrm{T}}(x^* + \rho e_j)| \leq b_i - \varepsilon + \frac{\varepsilon}{2} < b_i.$$

The points $x^{(0)} = x^*$, $x^{(j)} = x^* + \rho e_j$, together with the identity map $\pi\colon [n] \to [n]$, satisfy the statement.

We now consider the general case, with $d \geq 1$ (the case $d = 0$ is trivial). Since $\dim P = d$, there exists $I \subseteq [N_1]$ with $|I| \geq n - d$ such that the set

$$P^* := \{x \in \mathbb{R}^n \mid a_i^{\mathrm{T}} x = b_i \text{ for } i \in I \text{ and } a_i^{\mathrm{T}} x < b_i \text{ for } i \notin I\}$$

is included in P and has dimension d. After eliminating redundant equalities, if necessary, we can assume $|I| = n - d$.

In what follows, if $x \in \mathbb{R}^n$ and $J \subseteq [n]$, we denote by x_J the vector obtained by removing from x the coordinates with index not in J. Write $\bar{J} = [n] - J$. Let H be the affine linear variety given by $a_i^{\mathrm{T}} x = b_i$ for $i \in I$. Since $\dim H = d$ there exists $J \subseteq [n]$, $|J| = d$, such that we can express the coordinates x_j, $j \notin J$, in terms of the coordinates x_j, $j \in J$. More precisely, there exist $S \in \mathbb{R}^{\bar{J} \times J}$ and $c \in \mathbb{R}^{\bar{J}}$ such that

$$x_{\bar{J}} = S x_J + c. \tag{5.1}$$

The projection of P^* on $\mathbb{R}^J$ has dimension d. It is given by the set of strict inequalities obtained by substituting $x_{\bar{J}}$ in the inequalities $a_i x < b_i$, $i \notin I$, according to (5.1).

We now apply the full-dimensional case to this set of inequalities to find points $x_J^{(\ell)} \in \mathbb{R}^J$, $\ell = 0, \ldots, d$, satisfying the statement in $\mathbb{R}^J$ (for some bijection $\pi\colon [d] \to J$). Finally, we lift these points to $x^{(\ell)} \in \mathbb{R}^n$, $\ell \in [d]$, by using (5.1). ∎

Proof of Theorem 5.1. The hardness is immediate since adding d dummy variables reduces circuit satisfiability $\mathrm{CSAT}_{\mathrm{add}}$ to $\mathrm{DIM}_{\mathrm{add}}(d)$. (Here we use the convention $\dim \emptyset = -1$.)

For the membership, note that leaf sets are convex. Therefore, if such a leaf set contains the vertices of a d-dimensional simplex in $\mathbb{R}^n$, it contains the whole simplex. This ensures the correctness of the following algorithm for $\mathrm{DIM}_{\mathrm{add}}(d)$.

input $\mathcal{C}$
`compute` n`, the number of input gates of` $\mathcal{C}$
`guess an accepting leaf` ν`,` $x^{(0)}, \ldots, x^{(d)} \in \mathbb{R}^n$`,` $\rho \in \mathbb{R}$`, and` $\pi\colon [d] \to [n]$
`check` whether $x^{(0)}, \ldots, x^{(d)}$, ρ, and π satisfy the statement of Proposition 5.2
if $x^{(0)}, \ldots, x^{(d)}$ reach the leaf ν **then** `ACCEPT` **else** `REJECT`

This is clearly an $\mathrm{NP}_{\mathrm{add}}$-algorithm. ∎

5.2. Counting connected components. Consider the *reachability problem* $\mathrm{REACH}_{\mathrm{add}}$ to decide for a given additive circuit $\mathcal{C}$ and two points s and t whether these points are in the same connected component of the semilinear set X defined by $\mathcal{C}$. The corresponding counting problem $\#^{\mathsf{cc}}\mathrm{CSAT}_{\mathrm{add}}$ is the problem of counting the number of connected components of X given by $\mathcal{C}$.

The main result of this section is the following.

THEOREM 5.3. *The problems* $\mathrm{REACH}_{\mathrm{add}}$ *and* $\#^{\mathsf{cc}}\mathrm{CSAT}_{\mathrm{add}}$ *are* $\mathrm{PAR}_{\mathrm{add}}$*-complete and* $\mathrm{FPAR}_{\mathrm{add}}$*-complete with respect to Turing reductions, respectively.*

This result is inspired by an early paper by Reif [35] (see also [36]), which showed the PSPACE-hardness of a generalized movers problem in robotics. Reif's result implies that the analogue of $\mathrm{REACH}_{\mathrm{add}}$ for semialgebraic sets given by inequalities of (nonlinear) rational polynomials is PSPACE-hard. We cannot apply this result in our context, since we are dealing here with linear polynomials. However, we borrow from Reif's proof the idea to describe PSPACE by symmetric polynomial space Turing machines; see section 5.2.1.

The problem $\mathrm{REACH}_{\mathrm{add}}$ has a similar flavor as the undirected reachability problem for succinctly represented graphs, which asks whether two nodes s, t of such a graph G are connected by a path. By a *succinct representation* of a graph [18] we understand a Boolean circuit which decides, for a pair of nodes given in binary encoding, whether they are adjacent. This representation allows a polynomial size representation of graphs with exponentially many nodes. It is known that the undirected reachability problem for succinctly represented graphs is PSPACE-complete. A detailed treatment of the complexity of succinct problems can be found in [1].

The rest of this section is devoted to the proof of Theorem 5.3. It is organized as follows: after two preparatory subsections on symmetric Turing machines and embedding graphs, we provide the lower bound part of the proof in section 5.2.3. The upper bound part of the proof is given in section 5.2.4.

5.2.1. Symmetric Turing machines. Roughly speaking, a symmetric Turing machine [26] is a nondeterministic Turing machine with the property that its transition relation is symmetric. Thus its configuration digraph is in fact a graph, which is essential for capturing the symmetric reachability relation of $\mathrm{REACH}_{\mathrm{add}}$.

We briefly recall the notions which are essential for our proof. A (one tape) *symmetric Turing machine* M is given by $(Q, \Sigma, \Sigma_0, s, t, \Delta)$, where Q is a finite state space, $s \in Q$ is the initial state, $t \in Q$ is the final state, and Δ is a finite set of transitions. We will assume that the input alphabet Σ_0 equals $\{0, 1\}$ and that the machine alphabet Σ contains $0, 1$, and the blank "♭." The transitions $\delta \in \Delta$ are either of the form $\delta_1 = (p, a, 0, b, q)$ or $\delta_2 = (p, ab, cd, q)$, where $p, q \in Q$ and $a, b, c, d \in \Sigma$. The transition δ_1 is to be interpreted as follows: If the current state of the machine is p and the head of the machine is above a cell containing the symbol a, then the machine may rewrite this symbol by b and enter the state q without moving the head. Similarly, reading backwards, if the machine is in state q and the head of the machine is above a cell containing the symbol b, then the machine may rewrite this symbol by a and enter the state p without moving the head. The interpretation of the transition δ_2 is as follows: If the current state is p, the head of the machine is above a cell containing a, and the symbol in the cell to the immediate right is b, then the machine may rewrite a by c and b by d, move one step to the right, and enter the state q. The transition δ_2 may also be read backwards with the obvious interpretation. A *configuration* of M is an element $(q, h, w) \in Q \times \mathbb{N} \times \Sigma^{\mathbb{N}}$, where q is the current state, h the position of the head, and w the current tape contents. (All but finitely many components of w are blanks.) The transitions induce a symmetric relation on the set of configurations, which defines the (undirected) configuration graph.

In [26, Theorem 1] it is shown that any language recognized by a deterministic Turing machine may be recognized by a symmetric Turing machine respecting the same space bound.

5.2.2. Embedding graphs by straight-line segments. We first define two types of products of graphs. Then we study embeddings of such graph products in Euclidean space such that point location in these embedded graphs can be done by additive circuits. This will be needed for the lower bound proof in section 5.2.3.

DEFINITION 5.4. *Let $G_i = (V_i, E_i)$ be graphs, where $1 \le i \le t$.*

1. *The* product *$G_1 \times \cdots \times G_t$ is defined as a graph with the set of nodes $V_1 \times \cdots \times V_t$; two distinct nodes $u = (u_1, \ldots, u_t)$ and $v = (v_1, \ldots, v_t)$ are adjacent iff there exists i such that $\{u_i, v_i\} \in E_i$ and $u_j = v_j$ for all $j \neq i$. If $G_i = G$ for all i we will write G^t instead of $G \times \cdots \times G$.*

2. *The* extended product $G_1 \otimes \cdots \otimes G_t$ *also has set of nodes* $V_1 \times \cdots \times V_t$. *Now two distinct nodes* $u = (u_1, \ldots, u_t)$ *and* $v = (v_1, \ldots, v_t)$ *are adjacent iff for all* i *either* $u_i = v_i$ *or* $\{u_i, v_i\} \in E_i$.

Let $G = (V, E)$ be a graph and $\varphi\colon V \to \mathbb{R}^n$ be an injective map. We assign to each edge $e = \{u, v\}$ the closed (straight-line) segment $\varphi(e) := \{t\varphi(u) + (1-t)\varphi(v) \mid t \in [0,1]\}$.

DEFINITION 5.5. *The injective map* $\varphi\colon V \to \mathbb{R}^n$ induces an embedding *of the graph* $G = (V, E)$ *into* $\mathbb{R}^n$ (*by straight-line segments*) *iff*

$$\forall v \in V\ \forall e, e' \in E\ \big(\varphi(v) \in \varphi(e) \Longrightarrow v \in e\big) \wedge \big(\varphi(e) \cap \varphi(e') \neq \emptyset \Longrightarrow e \cap e' \neq \emptyset\big).$$

The edge skeleton *of the embedding is defined as the union of the segments corresponding to all edges.*

We denote by K_m the complete graph on the set of nodes $[m]$ and embed it in $\mathbb{R}^m$ by sending the node i to the ith canonical basis vector e_i. Thus we interpret K_m as the standard simplex in $\mathbb{R}^m$.

LEMMA 5.6.

1. *Assume that* $\varphi_i\colon V_i \to \mathbb{R}^{n_i}$ *induces an embedding of* G_i *for* $i \in [t]$. *Then* $V_1 \times \cdots \times V_t \to \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_t}, (v_1 \ldots, v_t) \mapsto (\varphi_1(v_1), \ldots, \varphi_t(v_t))$ *induces an embedding of* $G_1 \times \cdots \times G_t$.

2. *Let* $\varphi\colon V \to \mathbb{R}^n - \{0\}$ *induce an embedding of* $G = (V, E)$ *and* $m \in \mathbb{N}, m > 0$. *Assume further that* $\varphi(u), \varphi(v)$ *are linearly independent for all edges* $\{u, v\} \in E$. *Define* $\psi\colon [m] \times V \to (\mathbb{R}^n)^m$ *by* $\psi(i, v) := (0, \ldots, 0, \varphi(v), 0, \ldots, 0)$ *with* 0 *everywhere except at position* i. *Then* ψ *induces an embedding of the extended product* $K_m \otimes G$ *into* $(\mathbb{R}^n)^m$.

Proof. 1. This can be shown by straightforward verification.

2. The proof is a bit cumbersome, since it requires several case distinctions. Assume for simplicity that $m = 2$. Let $e = \{(1, u), (2, v)\}$, $e' = \{(1, u'), (2, v')\}$ be edges of $K_2 \otimes G$ such that $\psi(e) \cap \psi(e') \neq \emptyset$. Hence there exist $s, t \in [0, 1]$ such that $t\psi(1, u) + (1-t)\psi(2, v) = s\psi(1, u') + (1-s)\psi(2, v')$. This means that $t\varphi(u) = s\varphi(u')$ and $(1-t)\varphi(v) = (1-s)\varphi(v')$. We claim that $e \cap e' \neq \emptyset$.

If $t = 0$, then $s = 0$ (since $\mathrm{im}\varphi \subseteq \mathbb{R}^n - \{0\}$); hence $\varphi(v) = \varphi(v')$. Therefore $v = v'$, and we are done. Similarly, $t = 1$ implies $u = u'$. We may therefore assume that $s, t \notin \{0, 1\}$.

Suppose $u = v'$. Then either $u' = u$ or $\{u', u\} \in E$. In the latter case, $\varphi(u'), \varphi(u)$ are not linearly independent by assumption, which contradicts $t\varphi(u) = s\varphi(u')$. We may therefore assume that $u \neq v'$ and $v \neq u'$.

Since $s = t$ implies $u' = u$, we assume w.l.o.g. that $0 < s < t < 1$. It is easy to see that under these assumptions we have $\varphi(\{u, v\}) \cap \varphi(\{u', v'\}) \neq \emptyset$ (cf. Figure 5.1). As φ is an embedding, we deduce from this $\{u, v\} \cap \{u', v'\} \neq \emptyset$; hence $u = u'$ or $v = v'$, which proves the claim.

The other cases to be treated are simpler and are left to the reader. If $m > 2$, one can argue similarly. ∎

For positive integers a, b, p we define the graphs $G^p_{a,b} := K_a \otimes (K_b)^p$. Interpreting complete graphs as embedded via the standard simplices in Euclidean space and using the construction of the previous lemma, we get an embedding $\psi^p_{a,b}$ of $G^p_{a,b}$ in $\mathbb{R}^{abp}$. We denote the induced set of nodes in $\mathbb{R}^{abp}$ by R and the edge skeleton by T (omitting indices). The following property will be crucial.

LEMMA 5.7. *There are additive circuits of size polynomial in* a, b, p *depending uniformly on these parameters and performing the following tasks:*

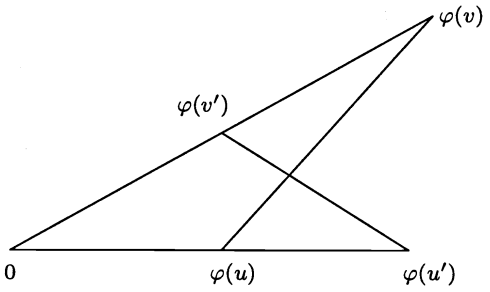


FIG. 5.1. *The segments $\varphi(\{u,v\})$ and $\varphi(\{u',v'\})$ intersect.*

1. *Compute the map $\psi^p_{a,b}\colon [a]\times[b]^p \to R$ and its inverse.*

2. *Decide in which set of the partition $\mathbb{R}^{abp} = (\mathbb{R}^{abp}\setminus T)\cup(T\setminus R)\cup R$ a given point in $\mathbb{R}^{abp}$ lies.*

3. *Compute the end points of the (unique) edge segment of $G^p_{a,b}$ in which a given point $x \in T\setminus R$ lies.*

Proof. To compute $\psi^p_{a,b}$ is straightforward. For its inverse, as well as for parts (2) and (3), note that the relevant information can be inferred from the signs of the components x_i of a given point in $x \in \mathbb{R}^{abp}$. No multiplications, not even scalar ones, are needed to perform these tasks. □

5.2.3. Lower bound for reachability. We are going to show the $\mathrm{PAR}_{\mathrm{add}}$-hardness of $\mathrm{REACH}_{\mathrm{add}}$ and $\#^{\mathsf{cc}}\mathrm{CSAT}_{\mathrm{add}}$. The next lemma tells us that it is sufficient to do this for $\mathrm{REACH}_{\mathrm{add}}$.

LEMMA 5.8. *The problem* $\mathrm{REACH}_{\mathrm{add}}$ *Turing reduces to* $\#^{\mathsf{cc}}\mathrm{CSAT}_{\mathrm{add}}$.

Proof. Let $X \subseteq \mathbb{R}^n$ be given by an additive circuit $\mathcal{C}$ accepting X, and suppose $s,t \in X$. Consider the following subset X' of $\mathbb{R}^{n+1}$:

$$X' := (X\times\{0\})\cup(\{s\}\times[0,1])\cup(\{t\}\times[0,1])\cup(\mathbb{R}^n\times\{1\}).$$

There is an $\mathrm{FP}_{\mathrm{add}}$-machine, which takes as input a circuit $\mathcal{C}$ together with $s,t\in\mathbb{R}^n$ and outputs an additive circuit $\mathcal{C}''$ deciding membership to X'. It is easy to check that s and t are connected in X iff X' has the same number of connected components as X (and one less otherwise). The latter condition can be tested by querying a $\#^{\mathsf{cc}}\mathrm{CSAT}_{\mathrm{add}}$-oracle twice, once with $\mathcal{C}$ and once with $\mathcal{C}''$. □

PROPOSITION 5.9. *The problem* $\mathrm{REACH}_{\mathrm{add}}$ *is* $\mathrm{PAR}_{\mathrm{add}}$*-hard with respect to Turing reductions.*

Proof. By Theorem 4.1(3) it is sufficient to prove that $\mathrm{REACH}_{\mathrm{add}}$ is PSPACE-hard. Thus let $L \subseteq \{0,1\}^\infty$ be any language in PSPACE. Let M be a symmetric Turing machine deciding membership in L with polynomial space bound function $p(n)$ (cf. section 5.2.1). For fixed input length n let H'_n denote the restriction of the configuration graph of M to the set of nodes $V_n := Q\times[p(n)]\times\Sigma^{p(n)}$. To an input $w\in\{0,1\}^n$ we assign the initial configuration $i(w) := (s,1,(w_1,\dots,w_n,\flat,\dots,\flat))$. We may assume w.l.o.g. that there is exactly one accepting configuration $f := (t,1,(\flat,\dots,\flat))$. Of course, the cardinality of V_n is exponential in n. However, it is clear that the graph H'_n can be succinctly represented by Boolean circuits of size polynomial in n.

Note that if $(q,h,w),(q',h',w')\in V_n$ are two configurations adjacent in H'_n, then $|h-h'|\le 1$, and the Hamming distance of w and w' is at most two. Let $\delta_2 = (p,ab,cd,q)$ be the transition between these configurations and w.l.o.g. $h' = h+1$.

We introduce an additional node $(\delta, h, \tilde{w})$, where $\tilde{w}$ is obtained from w by changing the hth entry to the one of w'. Thus the Hamming distances of both $w, \tilde{w}$ and $\tilde{w}, w'$ are at most one. We think of the node $(\delta, h, \tilde{w})$ as lying in between the nodes (q, h, w) and (q', h', w'). By this construction we obtain a *modified configuration graph* H_n on the set of nodes $V'_n := \tilde{Q} \times [p(n)] \times \Sigma^{p(n)}$ with the enlarged set of states $\tilde{Q} := Q \cup \Delta$. (Recall that Δ is the set of transitions of M.) If (q, h, w) and (q', h', w') are adjacent in H_n, then the Hamming distance of w and w' is at most one. Note that the graph H_n can also be succinctly represented by Boolean circuits of size polynomial in n.

By enumerating symbols we may assume that $\Sigma = [b]$ and $\tilde{Q} \times [p(n)] = [a(n)]$ with a polynomial function $a(n)$. From the above observations we conclude that the modified configuration graph H_n is a subgraph of the graph

$$G_n := G^{p(n)}_{a(n),b} := K_{a(n)} \otimes (K_b)^{p(n)}$$

defined in section 5.2.2. We embed G_n in $\mathbb{R}^{a(n)p(n)b}$ with the construction of section 5.2.2 using a map $\psi_n \colon \tilde{Q} \times [p(n)] \times \Sigma^{p(n)} \to R_n$ with induced set of nodes R_n. We denote the edge skeleton of this embedding by T_n and denote by S_n the edge skeleton of the induced embedding of the subgraph H_n of G_n.

By Lemma 5.7, membership in T_n can be decided by a uniform family of additive circuits of size polynomial in n. It is now easy to see that also membership in S_n can be decided by such a family $(\mathcal{C}_n)$ of additive circuits. In fact, we first find out whether a given point $x \in \mathbb{R}^{a(n)p(n)b}$ lies in R_n or T_n. If x lies in $T_n \setminus R_n$, then we compute the end points y, z of the (unique) edge segment of G_n in which x lies. Furthermore, we compute the inverse images $\eta := \psi_n^{-1}(y)$ and $\zeta := \psi_n^{-1}(z)$. Due to Lemma 5.7 all this can be done by a uniform family of additive circuits of size polynomial in n. Note that x lies in S_n iff η and ζ are adjacent in the modified configuration graph H_n. Since the latter can be succinctly described by Boolean circuits of size polynomial in n we can test this in polynomial time.

Consider the map φ associating $w \in \{0,1\}^n$ to $(\mathcal{C}_n, \psi_n(i(w)), \psi_n(f))$. By construction, $w \in L$ iff the configurations $i(w)$ and f can be connected in the modified configuration graph H_n. This in turn is equivalent to the statement that the points $\psi_n(i(w))$ and $\psi_n(f)$ are connected by a path in the skeleton S_n defined by $\mathcal{C}_n$. Therefore, φ is a reduction from L to $\mathrm{REACH}_{\mathrm{add}}$.

In addition, it is obvious that the additive circuit $\mathcal{C}_n$ for S_n as well as the points $\psi_n(i(w))$ and $\psi_n(f)$ can be computed in polynomial time from w. This completes the proof of the proposition. □

Remark 5.10. From the above proof it follows immediately that the problems $\mathrm{REACH}_{\mathrm{add}}$ and $\#^{\mathrm{cc}}\mathrm{CSAT}_{\mathrm{add}}$ restricted to closed input sets X remain complete for $\mathrm{PAR}_{\mathrm{add}}$ and $\mathrm{FPAR}_{\mathrm{add}}$, respectively.

5.2.4. Upper bound for reachability. In order to finish the proof of Theorem 5.3, it remains to show the following lemma.

LEMMA 5.11. *The problem* $\mathrm{REACH}_{\mathrm{add}}$ *is contained in* $\mathrm{PAR}_{\mathrm{add}}$, *and the problem* $\#^{\mathrm{cc}}\mathrm{CSAT}_{\mathrm{add}}$ *is contained* $\mathrm{FPAR}_{\mathrm{add}}$.

Proof. Let $\mathcal{C}$ be an additive circuit defining a set X. The computation of $\mathcal{C}$ can be unwound to a linear decision tree. We define a graph G, whose nodes consist of the accepting leaves of this tree and whose edges join two different leaves μ and ν iff the corresponding leaf sets D_μ and D_ν touch each other, that is, $\overline{D_\mu} \cap D_\nu \neq \emptyset$ or $D_\mu \cap \overline{D_\nu} \neq \emptyset$.

Let $K_1, \ldots, K_t$ be the connected components of the graph G. It is easy to see that X has exactly t connected components, namely the sets of the form $\bigcup_{\nu \in K_i} D_\nu$ for $i \in$

$[t]$. (Use the fact that leaf sets are convex and thus connected.) Therefore, the number of connected components of X is equal to the number of connected components of the graph G.

Of course, the graph G may be exponentially large. However, it can be represented by a weaker variant of the succinct representation discussed before. The nodes of G can be encoded by a word in $\{0,1\}^\infty$ encoding the corresponding path in the tree (0 means branching to the left, 1 means branching to the right). For two such given nodes μ, ν, we can decide in $\mathrm{NP}_{\mathrm{add}}$ whether they are connected in G by guessing a point $x \in \mathbb{R}^n$ and checking whether $x \in \overline{D_\mu} \cap D_\nu$ or $x \in D_\mu \cap \overline{D_\nu}$. The latter can be done as follows: we can easily write down the linear functions computed along the path of μ, thus obtaining a description of D_μ by a system of linear inequalities. A system describing the closure $\overline{D}_\mu$ can be obtained from the one describing D_μ by relaxing the occurring inequalities $<$ to $\leq$.

As in the proof of Savage's theorem we can decide in $\mathrm{PAR}_{\mathrm{add}}$ whether two nodes of G are connected by a path as follows. Let the predicate $\mathrm{PATH}(\mu, \nu, i)$ express that the nodes μ and ν are connected by a path of length at most 2^i. Then we implement $\mathrm{PATH}(\mu, \nu, i)$ by the recursive algorithm

`for all nodes` ω `test whether` $\mathrm{PATH}(\mu, \omega, i-1)$ `and` $\mathrm{PATH}(\omega, \nu, i-1)$

using only polynomial space (compare [34, p. 149]). By applying this procedure for every pair of nodes of G we can compute the number of connected components of G and thus that of X in $\mathrm{FPAR}_{\mathrm{add}}$. □

Remark 5.12. Let p be a prime. Then the problem of counting the number of connected components modulo p of a semilinear set given by an additive circuit is also $\mathrm{FPAR}_{\mathrm{add}}$-complete with respect to Turing reductions. For showing this, we have only to observe that the proof of Lemma 5.8 immediately extends to counting mod p.

5.3. Euler characteristic and Betti numbers. The main results of this section are the completeness results for $\mathrm{EULER}_{\mathrm{add}}$ and $\mathrm{BETTI}_{\mathrm{add}}(k)$ treated in section 5.3.4 and section 5.3.5. The following subsections prepare for the proofs.

5.3.1. Cell complexes and homology. We recall some notions from algebraic topology [20, 33]. A *cell* of dimension k is a topological space homeomorphic to the open k-dimensional unit ball $\mathrm{int}(B^k) := \{x \in \mathbb{R}^k \mid x_1^2 + \cdots x_k^2 < 1\}$. The closed unit ball will be denoted by B^k. Its boundary ∂B^k is homeomorphic to the $(k-1)$-dimensional unit sphere.

Assume that a topological Hausdorff space X is decomposed into a finite, disjoint union of cells: $X = \cup_{i=1}^N F_i$. The *k-skeleton* X^k is then defined as the union of the cells of dimension at most k. The cell decomposition is called a *finite cell complex* iff each cell F_i has a characteristic map, that is, a continuous map $h_i \colon B^k \to X$ mapping the boundary ∂B^k to X^{k-1} and such that h_i induces a homeomorphism of $\mathrm{int}(B^k)$ with F_i. In this case, X is necessarily compact.

In the following, we assume that X is a compact, semilinear subset of $\mathbb{R}^n$ decomposed into subsets F_i, $i \in [N]$, each described by a system

$$f_1 = a_1, \ldots, f_r = a_r,\ g_1 > d_1, \ldots, g_s > d_s,$$

where f_i, g_j are linear forms. Note that the F_i are bounded convex sets, which are open in their affine closure $\mathrm{aff}(F_i)$. In particular, each F_i is a cell, and ∂F_i is homeomorphic to a sphere. It is easy to see that this cell decomposition of X is a finite cell complex if the following *boundary condition* is satisfied:

$$\forall i, j \in [N], \quad F_i \cap \partial F_j \neq \emptyset \Longrightarrow F_i \subseteq \partial F_j. \tag{5.2}$$

This condition is equivalent to saying that the boundary ∂F_i of each cell is a union of cells. Such cell complexes will be called *semilinear cell complexes* in the sequel.

The following fact is well known [20, 33]. Let X be decomposed as a semilinear cell complex and denote by c_k the number of the k-cells of this decomposition. Then the Euler characteristic $\chi(X)$ can be computed as $\chi(X) = \sum_{k=0}^{n}(-1)^k c_k$.

We remark that the decomposition into leaf sets given by a ternary additive circuit may violate the boundary condition, which is a source of complications for our investigations. (For a definition of ternary circuits, see section 5.3.3.) For instance, consider the triangle X decomposed into the vertices $a := (0,0), b := (2,0), c := (0,2)$ and the open segments segments joining a with b, a with c, b with c, and a with $(1,1)$. Then the boundary point $(1,1)$ of the open segment joining a with $(1,1)$ is not a vertex.

In order to define the cellular homology of a semilinear cell complex, we first need to recall some facts about orientation.

Recall that an ordered basis of a finite-dimensional real vector space defines an *orientation* of this space. Two ordered bases are said to have the same orientation iff the transformation matrix sending one basis to the other has positive determinant. By an orientation of an affine linear subspace $A \subseteq \mathbb{R}^n$ we understand an orientation of its associated linear space $L(A)$. An orientation of a convex subset F of $\mathbb{R}^n$ is defined as an orientation of its affine hull $\mathrm{aff}(F)$. It will be convenient to write $L(F) := L(\mathrm{aff}(F))$.

Let $A \subseteq \mathbb{R}^n$ be given as the zero set of linear polynomials $f_1, \ldots, f_{n-k}$ in this order. Extend the sequence $f_1 - f_1(0), \ldots, f_{n-k} - f_{n-k}(0)$ to a basis of the space of linear forms such that the corresponding dual basis $(v_1, \ldots, v_n)$ is a positively oriented basis of $\mathbb{R}^n$. Then $(v_{n-k+1}, \ldots, v_n)$ is a basis of the linear space $L(A)$, which we define to be positively oriented. We will say that this is the orientation of A *induced by* $f_1, \ldots, f_{n-k}$. (Note that this is well defined.)

Let now A be the convex hull of a convex subset F of $\mathbb{R}^n$, and assume that H is a supporting hyperplane of F in A. That is, F lies on one side of H and the closure of F meets H. Then an *orientation of F induces an orientation of H* as follows: let y be a vector pointing from H outward of F. Then we say that a basis $v_1, \ldots, v_{n-1}$ of $L(H)$ is positively oriented with respect to the induced orientation iff $y, v_1, \ldots, v_{n-1}$ is a positively oriented basis of $L(A)$. (This is again well defined.)

Let now $X = \cup_{i=1}^{N} F_i$ be a semilinear cell complex, and assume that all the cells F_i are oriented. Let Φ_k denote the set of k-cells, and consider $F' \in \Phi_k$ and $F \in \Phi_{k+1}$. Assume that F' is contained in the closure of F. Then the affine hull of F' is a hyperplane in the affine hull of F supporting the convex set F. Therefore, the orientation of F induces an orientation on F' as explained above.

We define the *incidence number* $[F, F']$ of $F \in \Phi_{k+1}$ and $F' \in \Phi_k$ by

$$[F, F'] = \begin{cases} 0 & \text{iff } F' \text{ is not contained in the closure of } F, \\ 1 & \text{if the orientation } F \text{ induces on } F' \text{ is the same as that of } F', \\ -1 & \text{otherwise.} \end{cases}$$

For $k \geq 0$ the *incidence matrix* I_k is the matrix associated with $I_k \colon \Phi_{k+1} \times \Phi_k \to \mathbb{Z}$ by $I_k(F, F') = [F, F']$.

Let $\mathcal{C}_k$ be the $\mathbb{Q}$-vector space having Φ_k as a basis. The boundary map $\partial_k \colon \mathcal{C}_{k+1} \to \mathcal{C}_k$ is the $\mathbb{Q}$-linear map defined for $F \in \Phi_{k+1}$ by

$$\partial_k(F) = \sum_{F' \in \Phi_k} [F, F']\, F'.$$

The image $B_k := \mathrm{im}\partial_k$ of ∂_k is called the vector space of k-boundaries, and the kernel $Z_k := \ker \partial_{k-1}$ is called the space of k-cycles. The kth *cellular homology vector space* is defined as $H_k := Z_k/B_k$. It is well known [20, 33] that H_k is isomorphic to the singular homology vector space $H_k(X;\mathbb{Q})$. Therefore, $b_k(X) := \dim H_k$ is the kth Betti number, which is independent of the cell decomposition and of the choice of orientations for its cells. We have $b_k(X) = \dim Z_k - \dim B_k = c_k - \rho_{k-1} - \rho_k$, where $\rho_k := \mathrm{rank}\,\partial_k$ and $c_k = |\Phi_k|$.

5.3.2. Reduction to the compact case. The technical result developed in this section will be needed in the upper bound proofs for the problems $\mathrm{EULER}_{\mathrm{add}}$ and $\mathrm{BETTI}_{\mathrm{add}}(k)$. The purpose is to show that the closedness assumption on X can be strengthened to compactness w.l.o.g.

Let us first show that both closedness and compactness of a semilinear set can be checked within the allowed resources, that is, in additive polynomial time with access to a #P-oracle. We write $\|x\|_\infty := \max_{i\le n} |x_i|$ for $x \in \mathbb{R}^n$.

LEMMA 5.13. *Both closedness and compactness of a set X given by an additive circuit can be decided in $\mathrm{P}_{\mathrm{add}}^{\#\mathsf{P}}$.*

Proof. The boundedness of $X \subseteq \mathbb{R}^n$ can be expressed as follows:

$$\exists R \in \mathbb{R}\ \forall x\ (x \in X \Longrightarrow \|x\|_\infty \le R).$$

Hence this property can be decided in Σ^2_{add}. The closedness of $X \subseteq \mathbb{R}^n$ can be expressed by

$$\forall y\ \exists \epsilon\ \forall x\ (y \notin X, x \in X, \epsilon > 0 \Longrightarrow \|x - y\|_\infty \ge \epsilon).$$

Hence this property can be decided in Π^3_{add}. Now use Corollary 4.6. □

We recall a further notion from topology [20, 33]. A subspace A of a space X is called a *strong deformation retract* of X if there is a continuous map $F\colon X \times [0,1] \to X$ such that $F(x,0) = x$, $F(x,1) \in A$, and $F(a,t) = a$ for all $x \in X$, $a \in A$, and $t \in [0,1]$. It is a well-known fact that if A is a strong deformation retract of X, then the inclusion of A in X induces an isomorphism of the homology vector spaces $H_k(A;\mathbb{Q}) \simeq H_k(X;\mathbb{Q})$. In particular, the spaces A and X have the same Betti numbers.

LEMMA 5.14. *Let $X \subseteq \mathbb{R}^n$ be a closed semilinear set given by an additive circuit $\mathcal{C}$. As usual, we denote by D_ν the corresponding leaf sets. Then*

1. *we can compute from $\mathcal{C}$ in $\mathrm{FP}_{\mathrm{add}}$ a real number $R > 0$ such that*

$$\forall \nu\ \big(D_\nu \ne \emptyset \Longrightarrow D_\nu \cap [-R,R]^n \ne \emptyset\big); \tag{5.3}$$

2. *if R satisfies (5.3), then $X_R := X \cap [-R,R]^n$ is a strong deformation retract of X.*

Proof. 1. Each leaf set D_ν is defined by a set of sign conditions for the values computed by the circuit. On an input $y \in \mathbb{R}^n$ these values are of the form

$$z = \sum_{i=1}^{n} a_i y_i + \sum_{i=1}^{k} b_i \alpha_i + c,$$

where $\alpha_1, \dots, \alpha_k$ are the constants of $\mathcal{C}$ and the coefficients a_i, b_i, c are integers of bit-size at most $s := \mathrm{size}(\mathcal{C})$. If D_ν is not empty, Theorem 2.6 implies that there is a point $y \in D_\nu$ such that $y_i = \sum_{j=1}^{k} u_{ij}\alpha_j + w_i$, where the u_{ij}, w_i are rationals of

bit-size at most $L := (sn)^c$, c being some universal constant. Hence $\max_i |y_i| \leq R$, where $R := 2^L\big(1 + \sum_{j=1}^k |\alpha_j|\big)$. Therefore, the bound R satisfies the condition (5.3). Moreover, it is clear that R can be computed in $\mathrm{FP}_{\mathrm{add}}$ from $\mathcal{C}$.

2. Assume w.l.o.g. that X is not compact. Consider the one-point compactification $\dot{X}$ of X, which is explicitly defined as follows. Let $S^n \subset \mathbb{R}^{n+1}$ be the n-dimensional sphere and $N = (0, 0, \ldots, 0, 1) \in S^n$ be its north pole. Projection from N yields a homeomorphism between $S^n - \{N\}$ and $\mathbb{R}^n \times \{-1\}$ and therefore a homeomorphism between $S^n - \{N\}$ and $\mathbb{R}^n$. The closure $\dot{X}$ of the image of X in S^n (which consists of attaching N to this image) is called a *one-point compactification* of X. The decomposition into leaf sets of X becomes a cell decomposition of $\dot{X}$, which can be turned into a finite cell complex by refinement. The claim is now a consequence of the following intuitive topological fact, whose formal proof is left to the reader. Let Y be a finite cell complex and p be a vertex of Y. Assume that U is an open neighborhood of p so small that $\{p\}$ is the only cell of the complex contained in U. Then $Y \setminus U$ is a strong deformation retract of $Y \setminus \{p\}$. □

5.3.3. Universal cell decompositions. We adopt the notation $\mathcal{F}_{s,n}$ for the universal cell decomposition for the parameters s, n, introduced in section 4.1.

LEMMA 5.15. *If $X \subseteq \mathbb{R}^n$ is compact and a finite union of faces in $\mathcal{F}_{s,n}$, then the decomposition of X is a semilinear cell complex.*

Proof. It is obvious that the boundary condition (5.2) is satisfied. □

Let $\mathcal{C}'$ be an additive circuit defining the semilinear set $X \subseteq \mathbb{R}^n$. At the price of at most doubling the size of the circuit, we can transform $\mathcal{C}'$ into a *ternary additive circuit* $\mathcal{C}$, which branches according to the sign of intermediate results in a ternary way ($< 0, = 0, > 0$) instead of branching in a binary way according to $x \geq 0$ or $x < 0$. Each (nonempty) leaf set D_ν of $\mathcal{C}$ is described in the form

$$f_1 = a_1, \ldots, f_r = a_r,\ g_1 > d_1, \ldots, g_s > d_s,$$

where the $f_i - a_i$ and $g_j - d_j$ are the linear polynomials computed along the path leading to the leaf ν. Note that the linear forms f_i, g_j have integer coefficients of bit-size at most 2^s, where s is the size of the circuit $\mathcal{C}$. If the circuit uses only the constants $0, 1$, then a_i, d_j are also integers of bit-size at most 2^s. In the general case, however, a_i, d_j are real numbers. In the first case, each leaf set D_ν is a union of faces of $\mathcal{F}_{s,n}$. Thus, $\{F \in \mathcal{F}_{s,n} \mid F \subseteq X\}$ is a refinement of the decomposition of X into the leaf sets. By Lemma 5.15 this decomposition is a semilinear cell complex if X is compact.

Let X be a compact finite union of cells of $\mathcal{F}_{s,n}$. To define (and compute) the cellular homology groups of X we need to fix orientations on the cells $F \in \mathcal{F}_{s,n}$. The cellular homology groups are independent of the chosen orientations, so we will make this choice in a convenient way as explained below.

By identifying a sequence $(f_1, \ldots, f_k)$ in $(\mathcal{H}_{s,n})^k$ with the sequence of coefficients of $f_1, \ldots, f_k$ (in a fixed order) and using the lexicographical ordering, we may consider $(\mathcal{H}_{s,n})^k$ as a totally ordered set. We can extend this order to the union $\mathcal{H}_{s,n}^\infty$ of the $(\mathcal{H}_{s,n})^k$, for $k \in \mathbb{N}$, by requiring that elements of $(\mathcal{H}_{s,n})^k$ are strictly smaller than elements of $(\mathcal{H}_{s,n})^{k'}$ for $k < k'$.

For $F \in \mathcal{F}_{s,n}$ let $(f_1, \ldots, f_{n-k})$ be the smallest sequence in $\mathcal{H}_{s,n}^\infty$ such that F is contained in the zero set of $f_1, \ldots, f_{n-k}$. Then $k = \dim F$. We define the orientation of F as the orientation of $\mathrm{aff}(F)$ induced by this smallest sequence $(f_1, \ldots, f_{n-k})$ (cf. section 5.3.1).

In the sequel, $\mathcal{F}$ shall denote the union of the $\mathcal{F}_{s,n}$, and $\mathcal{H}$ the union of the $\mathcal{H}_{s,n}$, over all $s, n \in \mathbb{N}$, respectively. Recall that we encode the faces $F \in \mathcal{F}$ by triples $(s, n, x) \in \mathbb{N}^2 \times \mathbb{Q}^n$ with a rational point $x \in F$ of bit-size at most $(sn)^c$. Let the *incidence function* $I\colon \mathcal{F} \times \mathcal{F} \to \{-1, 0, 1\}$ be defined by $I(F, F') := [F, F']$, if $F, F' \in \mathcal{F}_{s,n}$ for some s, n and $\dim F = \dim F' + 1$, and $I(F, F') = 0$ otherwise.

LEMMA 5.16.

1. *The membership decision problem* $\{(F, x) \in \mathcal{F} \times \mathbb{R}^\infty \mid x \in F\}$ *is in* $\mathrm{PH}^0_{\mathrm{add}}$.
2. *The containment decision problem* $\{(F, f) \in \mathcal{F} \times \mathcal{H} \mid F \subseteq Z(f)\}$ *is in* PH.

Here $Z(f)$ *denotes the zero set of the polynomials of the sequence* f.

3. *The closure containment problem* $\{(F', F) \in \mathcal{F} \times \mathcal{F} \mid F' \subseteq \partial F\}$ *is in* PH.
4. *There is a function* $\mathcal{F} \to \mathbb{Q}^\infty$ *in* $\mathsf{FP}^{\mathsf{PH}}$ *mapping* F *to a positively oriented basis of the linear space* $L(F)$ *associated with the affine hull of* F.
5. *The incidence function* $I\colon \mathcal{F} \times \mathcal{F} \to \{-1, 0, 1\}$ *can be computed in* $\mathsf{FP}^{\mathsf{PH}}$.

Proof. 1. Let $F \in \mathcal{F}_{s,n}$ be given by the rational point $x_0 \in F$. We have

$$x \in F \iff \forall f \in \mathcal{H}_{s,n}\ \big(\mathrm{sgn} f(x) = \mathrm{sgn} f(x_0)\big).$$

This condition is expressible in $(\Pi^1_{\mathrm{add}})^0 \subseteq \mathrm{PH}^0_{\mathrm{add}}$, which shows the claim.

2. Replacing in the statement for all x ($x \in F \Rightarrow x \in Z(f)$) the predicate "$x \in F$" according to part 1, we obtain a $(\Pi^1_{\mathrm{add}})^0$-statement. Since F and f are discrete, the containment decision problem even belongs to the Boolean part of $(\Pi^1_{\mathrm{add}})^0$ and thus to Π^1 by [14] (cf. Remark 4.12).

3. Using part 1, we can express "$x \in \overline{F}$" by a $(\Pi^2_{\mathrm{add}})^0$-statement. Hence $F' \subseteq \partial F$ can also be expressed by a $(\Pi^2_{\mathrm{add}})^0$-statement. Since F, F' are discrete, this statement is even in Π^2.

4. Using part 2, we see that the following condition is in Π^2: $(f_1, \ldots, f_{n-k})$ is the smallest sequence in $\mathcal{H}^\infty_{s,n}$ such that F is contained in the zero set of $f_1, \ldots, f_{n-k}$. The assertion follows now by Remark 2.11(ii).

5. For given $F, F' \in \mathcal{F}_{s,n}$ we first check in PH whether $F' \subseteq \partial F$ and $\dim F' = \dim F - 1$ using parts 3 and 4. Then we compute positively oriented bases $u_1, \ldots, u_{k+1}$ of $L(F)$ and $v_1, \ldots, v_k$ of $L(F')$ in $\mathsf{FP}^{\mathsf{PH}}$ according to part 4. If F and F' are represented by the rational points x and x', respectively, then $y := x' - x$ is a vector in $L(F)$ pointing from $L(F')$ outside of F. The incidence number $[F, F']$ equals 1 iff the bases $y, v_1, \ldots, v_k$ and $u_1, \ldots, u_{k+1}$ have the same orientation, which can be checked in P. □

We extend now to the case when there are real constants what we have discussed before.

Let $s, n, \ell \in \mathbb{N}$ and $\eta \in \mathbb{R}^\ell$. By intersecting the universal cell decomposition of $\mathbb{R}^{n+\ell}$ for the parameters $s, n + \ell$ with the hyperplane $H(\eta) := \{(x, y) \in \mathbb{R}^{n+\ell} \mid y = \eta\}$, we get a cell decomposition of $\mathbb{R}^n \times \{\eta\}$, which we identify with $\mathbb{R}^n$. More specifically, each face $F \in \mathcal{F}_{s+\ell,n}$ induces a face $F(\eta)$ of this decomposition, defined by $F(\eta) \times \{\eta\} := F \cap H(\eta)$, provided this intersection is nonempty. Note that each $F(\eta)$ is defined by putting sign conditions on all polynomials of the form $a_0 + \sum_{j=1}^{\ell} b_j \eta_j + \sum_{i=1}^{n} a_i X_i$, where $\sum_{i=0}^{n} |a_i| + \sum_{j=1}^{\ell} |b_j| \le 2^s$. We write $\mathcal{F}_{s,n}(\eta) := \{F(\eta) \mid F \in \mathcal{F}_{s,n}\}$ and call this the *universal cell decomposition* of $\mathbb{R}^n$ for the parameters s, n and vector of constants $\eta \in \mathbb{R}^\ell$. Moreover, we write $\mathcal{F}(\eta)$ for the union of the $\mathcal{F}_{s,n}(\eta)$ over all $s, n \in \mathbb{N}$.

Most of the results shown so far in this subsection extend to this more general notion of universal cell decompositions in a natural way. For instance, Lemma 5.15

extends immediately to $\mathcal{F}_{s,n}(\eta)$. A face $F(\eta) \in \mathcal{F}(\eta)$ is encoded by $F \in \mathcal{F}$, which is itself encoded by a small rational point.

LEMMA 5.17. *An analogue of Lemma* 5.16 *holds, where the complexity classes* $\mathrm{PH}^0_{\mathrm{add}}$, FP, *and* PH *have to be replaced by* $\mathrm{PH}_{\mathrm{add}}$, $\mathsf{FP/poly}$, *and* $\mathsf{PH/poly}$, *respectively.*

The proof is a straightforward extension of the proof of Lemma 5.16. For instance, for treating the closure containment problem $\{(F', F) \in \mathcal{F}(\eta) \times \mathcal{F}(\eta) \mid F' \subseteq \partial F\}$ one first shows that this problem is in $\mathrm{PH}_{\mathrm{add}}$. Then, since this is a discrete problem, one concludes that it is in the Boolean part of $\mathrm{PH}_{\mathrm{add}}$ and thus in $\mathsf{PH/poly}$ by [14] (cf. Remark 4.12).

5.3.4. Euler characteristic. The Euler characteristic $\chi(X)$ is a fundamental invariant of a topological space.

Let $\mathrm{EULER}_{\mathrm{add}}$ denote the following problem: given an additive circuit $\mathcal{C}$ defining a *closed* semilinear set X, decide whether X is empty and, if not, compute its Euler characteristic $\chi(X)$. Hence only circuits defining closed semilinear sets are considered to be admissible inputs.

THEOREM 5.18. *The problem* $\mathrm{EULER}_{\mathrm{add}}$ *is* $\mathrm{FP}^{\#\mathrm{P}_{\mathrm{add}}}_{\mathrm{add}}$*-complete with respect to Turing reductions.*

Proof. We first show that $\mathrm{EULER}_{\mathrm{add}}$ is $\#\mathrm{P}_{\mathrm{add}}$-hard. Note that the problem $\mathrm{CSAT}_{\mathrm{add}}$ introduced in section 1.4 trivially reduces to $\mathrm{EULER}_{\mathrm{add}}$ by the definition of the latter problem. Hence $\mathrm{NP}_{\mathrm{add}} \subseteq \mathrm{P}^{\mathrm{EULER}_{\mathrm{add}}}_{\mathrm{add}}$. Therefore, by Theorem 5.1 we have $\mathrm{DIM}_{\mathrm{add}}(1) \in \mathrm{P}^{\mathrm{EULER}_{\mathrm{add}}}_{\mathrm{add}}$.

It is now easy to design a Turing reduction from $\#\mathrm{CSAT}_{\mathrm{add}}$ to $\mathrm{EULER}_{\mathrm{add}}$. On input of an additive circuit $\mathcal{C}$, first decide whether X is finite using oracle calls to $\mathrm{DIM}_{\mathrm{add}}(1)$, and hence to $\mathrm{EULER}_{\mathrm{add}}$. If no, return ∞; otherwise, return $\chi(X)$. Since $\#\mathrm{CSAT}_{\mathrm{add}}$ is $\#\mathrm{P}_{\mathrm{add}}$-complete by Theorem 3.6, this proves the hardness.

It remains to prove that $\mathrm{EULER}_{\mathrm{add}}$ is contained in $\mathrm{FP}^{\#\mathrm{P}_{\mathrm{add}}}_{\mathrm{add}}$. By Lemma 5.14, we may restrict our discussion to additive circuits defining a compact semilinear set $X \subseteq \mathbb{R}^n$. Assume that X is given by a ternary additive circuit $\mathcal{C}$ of size s using the real constants $\eta_1, \ldots, \eta_\ell$. Then each of the leaf sets of $\mathcal{C}$ is a union of faces in $\mathcal{F}_{s,n}(\eta)$. Hence the decomposition of X into the faces $F \in \mathcal{F}_{s,n}(\eta)$ contained in X is a semilinear cell complex. If $c_k(\mathcal{C})$ denotes the number of k-cells of this cell complex, we have $\chi(X) = \sum_{k=0}^{n} (-1)^k c_k(\mathcal{C})$.

Lemma 5.16 and its extension, Lemma 5.17, imply that on input $\mathcal{C}$ and $F \in \mathcal{F}_{s,n}(\eta)$ the property $F \in \Phi_k(\mathcal{C})$ can be tested in $\mathrm{DPH}_{\mathrm{add}}$. Therefore, $c_k(\mathcal{C})$ can be computed from $\mathcal{C}$ in $\mathrm{D}\#\cdot\mathrm{PH}_{\mathrm{add}}$, which is contained in $\mathrm{FP}^{\#\mathsf{P}}_{\mathrm{add}}$ by Corollary 4.6. This shows that $\mathrm{EULER}_{\mathrm{add}}$ belongs to $\mathrm{FP}^{\#\mathsf{P}}_{\mathrm{add}}$. ∎

5.3.5. Betti numbers. The kth Betti number $b_k(X)$ of a space X is defined as the dimension of the kth (singular) homology vector space $H_k(X; \mathbb{Q})$ ($k \in \mathbb{N}$). The Betti numbers modulo a prime p are defined by replacing the coefficient field $\mathbb{Q}$ by the finite field $\mathbb{F}_p$.

For $k \in \mathbb{N}$, we define $\mathrm{BETTI}_{\mathrm{add}}(k)$ to be the problem of computing the kth Betti number of a *closed* semilinear set given by an additive circuit. Recall that for $k = 0$ this is just the problem of counting the number of connected components. The problem of computing the kth Betti number modulo a prime p shall be denoted by $\mathrm{BETTI}_{\mathrm{add}}(k, \bmod p)$.

The goal of this section is the proof of the following result, extending Theorem 5.3.

THEOREM 5.19. *For any* $k \in \mathbb{N}$ *and any prime* p*, the problems* $\mathrm{BETTI}_{\mathrm{add}}(k)$ *and* $\mathrm{BETTI}_{\mathrm{add}}(k, \bmod p)$ *are* $\mathrm{FPAR}_{\mathrm{add}}$*-complete with respect to Turing reductions.*

The next lemma provides the lower bound part of the proof of Theorem 5.19.

LEMMA 5.20. $\mathrm{BETTI}_{\mathrm{add}}(k)$ *and* $\mathrm{BETTI}_{\mathrm{add}}(k, \mathrm{mod}\, p)$ *are* $\mathrm{FPAR}_{\mathrm{add}}$*-hard with respect to Turing reductions for any* $k \in \mathbb{N}$ *and any prime* p.

Proof. We will exhibit a Turing reduction from $\#^{\mathsf{cc}}\mathrm{CSAT}_{\mathrm{add}}$ to $\mathrm{BETTI}_{\mathrm{add}}(k)$. W.l.o.g., we assume $k > 0$.

The *suspension* $S(X)$ of a topological space X is defined as the space obtained from the cylinder $X \times [0,1]$ over X by identifying the points in each of the sets $X \times \{0\}$ and $X \times \{1\}$. Essentially, this is a double cone with basis X. It is well known that if $X \neq \emptyset$, the Betti numbers of $S(X)$ and X are related as follows (cf. [20, 33]):

$$b_{k+1}(S(X)) = \begin{cases} b_k(X) & \text{if } k > 0, \\ b_0(X) - 1 & \text{if } k = 0. \end{cases} \tag{5.4}$$

If $X \subseteq \mathbb{R}^n$ is given by an additive circuit, then we will use the following alternative definition for the suspension:

$$S_1(X) := (X \times [0,1]) \cup (\mathbb{R}^n \times \{0\}) \cup (\mathbb{R}^n \times \{1\}),$$

which, if $X \neq \emptyset$, is homotopy equivalent to $S(X)$ and therefore has the same Betti numbers. Also,

$$b_k(S_1(\emptyset)) = \begin{cases} 0 & \text{if } k > 0, \\ 2 & \text{if } k = 0. \end{cases} \tag{5.5}$$

This alternative definition of suspension has the advantage that it is easy to transform an additive circuit describing X to one describing $S_1(X)$. Note that $S_1(X)$ is closed if X is closed. If we iterate this construction $k+1$ times starting with $X \subseteq \mathbb{R}^n$, we get a set $S_{k+1}(X) \subseteq \mathbb{R}^{n+k+1}$, which satisfies, by (5.4) and (5.5),

$$b_k(S_{k+1}(X)) = b_0(S_1(X)) - 1 = \begin{cases} 0 & \text{if } X \neq \emptyset, \\ 1 & \text{if } X = \emptyset. \end{cases}$$

This allows one to decide whether $X = \emptyset$ by one query to $\mathrm{BETTI}_{\mathrm{add}}(k)$. If $X = \emptyset$, then we return 0. Otherwise, note that, by (5.4), $b_0(X) = b_k(S_k(X)) + 1$ (since $k > 0$), and we may return $b_0(X)$ after another query to $\mathrm{BETTI}_{\mathrm{add}}(k)$. Strictly speaking, this is a reduction from the restriction of $\#^{\mathsf{cc}}\mathrm{CSAT}_{\mathrm{add}}$ to closed input sets X. However, this is sufficient by Remark 5.10.

The same reduction can be made for the Betti numbers modulo a prime. The hardness follows in this case by appealing to Remark 5.12. □

Before proving the upper bound, we make a short digression on space efficient linear algebra. It is well known [32] that the rank of an $N \times N$ integer matrix A, whose entries have bit-size at most L, can be computed by uniform Boolean circuits with depth $(\log L + \log N)^{O(1)}$ (and similary for matrices over $\mathbb{F}_p$). Using Borodin's result [6], this can be translated to a polylogarithmic space computation of the rank of A by a Turing machine.

Similarly as for graphs, we will understand by a *succinct representation* of an integer matrix $A = (a_{ij})$ a Boolean circuit B computing the matrix entry a_{ij} from the index pair (i, j) given in binary. From the above discussion we conclude the following lemma.

LEMMA 5.21. *The rank of an* $N \times N$ *integer matrix* A *given in succinct representation by a Boolean circuit* B *can be computed by a Turing machine with space*

polynomial in $\log N$*, the depth of* B*, and the log of the maximal bit-size of the entries of* A*.*

We finish now the proof of Theorem 5.19 by showing the following upper bound.

PROPOSITION 5.22. $\mathrm{BETTI}_{\mathrm{add}}(k)$ *and* $\mathrm{BETTI}_{\mathrm{add}}(k, \mathrm{mod}\, p)$ *are both contained in* $\mathrm{FPAR}_{\mathrm{add}}$.

Proof. 1. We first prove that $\mathrm{BETTI}^0_{\mathrm{add}}(k)$ is in $\mathsf{FPSPACE}$. Let the closed semilinear set $X \subseteq \mathbb{R}^n$ be given by an additive circuit $\mathcal{C}'$, whose only constants are 0 and 1. By Lemma 5.14, we may assume w.l.o.g. that X is compact. We transform $\mathcal{C}'$ into a ternary additive circuit $\mathcal{C}$ of size s as in section 5.3.3. Consider the cell decomposition of X induced by $\mathcal{C}$. It is clear that each of its leaf sets is a union of faces in $\mathcal{F}_{s,n}$. Lemma 5.15 implies that the decomposition of X into the faces $F \in \mathcal{F}_{s,n}$ contained in X is a semilinear cell complex.

We put $\Phi_k(\mathcal{C}) := \{F \in \mathcal{F}_{s,n} \mid \dim F = k, F \subseteq X\}$ and $c_k(\mathcal{C}) := |\Phi_k(\mathcal{C})|$ for $k \in \mathbb{N}$. Let $\rho_k(\mathcal{C})$ denote the rank of the incidence matrix $I_k(\mathcal{C})\colon \Phi_{k+1}(\mathcal{C}) \times \Phi_k(\mathcal{C}) \to \mathbb{Z}, (F, F') \mapsto [F, F']$. In section 5.3.1 it was shown that the kth Betti number $b_k(X)$ of X can be expressed as $b_k(X) = c_k(\mathcal{C}) - \rho_{k-1}(\mathcal{C}) - \rho_k(\mathcal{C})$.

Lemma 5.16 implies that on input $\mathcal{C}$ and F the property $F \in \Phi_k(\mathcal{C})$ can be tested in PH. Therefore, $c_k(\mathcal{C})$ can be computed from $\mathcal{C}$ in $\# \cdot \mathsf{PH} \subseteq \mathsf{FPSPACE}$.

Thus it remains to show that for each $k \in \mathbb{N}$ the function $\mathcal{C} \mapsto \rho_k(\mathcal{C})$ can be computed in $\mathsf{FPSPACE}$. It follows from Lemma 5.16(5) that $I_k(\mathcal{C})(F, F')$ can be computed from $\mathcal{C}, F, F'$ in $\mathsf{FPSPACE}$. Hence, by Borodin's result [6] a succinct representation B of the incidence matrix $I_k(\mathcal{C})$ having depth polynomial in s can be computed from $\mathcal{C}$ in $\mathsf{FPSPACE}$. By Lemma 5.21, we can compute the rank $\rho_k(\mathcal{C})$ from B by a Turing machine in space polynomial in s. Altogether, we get a computation of $\rho_k(\mathcal{C})$ in $\mathsf{FPSPACE}$.

2. We now prove that $\mathrm{BETTI}_{\mathrm{add}}(k)$ belongs to $\mathrm{FPAR}_{\mathrm{add}}$. Assume that the compact semilinear set $X \subseteq \mathbb{R}^n$ is given by a ternary additive circuit $\mathcal{C}$ of size s using the real constants $\eta_1, \ldots, \eta_\ell$. Then each of the leaf sets of $\mathcal{C}$ is a union of faces in $\mathcal{F}_{s,n}(\eta)$. Hence the decomposition of X into the faces $F \in \mathcal{F}_{s,n}(\eta)$ contained in X is a semilinear cell complex. The rest of the argument is based on Lemma 5.17 and similar as before.

3. The case of positive characteristic can be settled similarly. □

COROLLARY 5.23. $\mathrm{BETTI}_{\mathrm{add}}(k)$ *Turing reduces to* $\mathrm{EULER}_{\mathrm{add}}$ *for some* $k \in \mathbb{N}$ *iff* $\mathsf{PSPACE} \subseteq \mathsf{P}^{\#\mathsf{P}}/\mathsf{poly}$*. It does so with a constant-free reduction iff* $\mathsf{PSPACE} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

Proof. This follows from Theorem 5.18, Theorem 5.19, and Corollary 4.11. □

5.3.6. Some completeness results in the Turing model. The results of section 4 and sections 5.1–5.3 can be combined to show completeness results for natural geometric problems in the discrete setting.

An additive circuit $\mathcal{C}$ whose only constants are 0 and 1 can be encoded in $\{0,1\}^\infty$. Thus, one may consider discrete versions $\mathrm{EULER}^0_{\mathrm{add}}$ and $\mathrm{BETTI}^0_{\mathrm{add}}(k)$, which are defined as $\mathrm{EULER}_{\mathrm{add}}$ and $\mathrm{BETTI}_{\mathrm{add}}(k)$, respectively, but are restricted to constant-free circuits. Note that since $\mathrm{BP}((\Pi^3_{\mathrm{add}})^0) = \Pi^3$ (cf. [14]), a corresponding version of Lemma 5.13 holds; i.e., closedness and compactness of a set given by a constant-free additive circuit can be decided in $\mathsf{P}^{\#\mathsf{P}}$.

For these discrete problems the following results hold.

COROLLARY 5.24. *The problems* $\mathrm{EULER}^0_{\mathrm{add}}$ *and* $\mathrm{BETTI}^0_{\mathrm{add}}(k)$*,* $k \in \mathbb{N}$*, are complete with respect to Turing reductions in* $\mathsf{FP}^{\#\mathsf{P}}$ *and* $\mathsf{FPSPACE}$*, respectively. A similar statement holds for the computation of Betti numbers modulo a prime.*

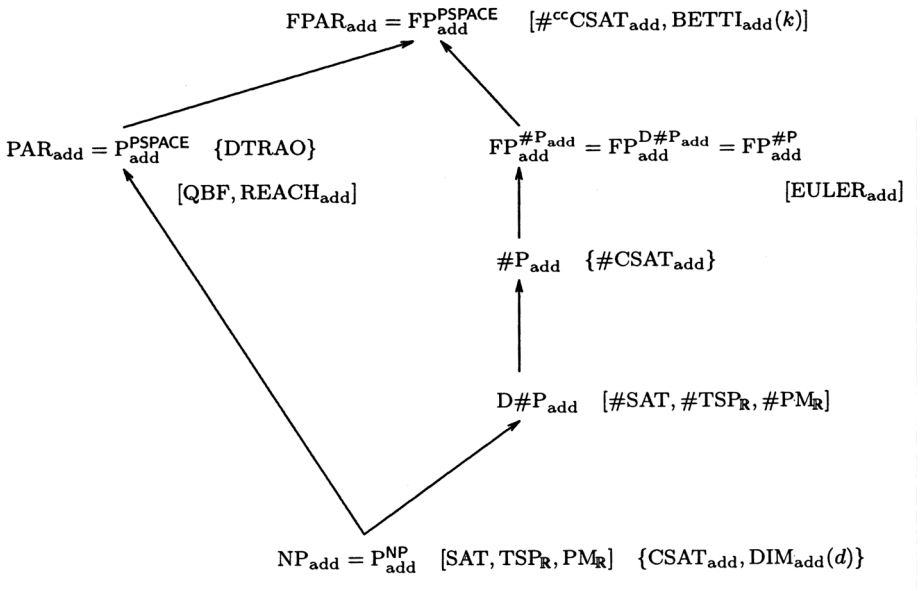


FIG. 6.1. *Survey of main results.*

Proof. This could be shown by checking in detail the proofs of Theorem 5.18 and Theorem 5.19. More elegantly, we can derive Corollary 5.24 from general principles using the concept of Boolean parts. We then need only to check that the reductions in the proofs of Theorem 3.6, Theorem 5.18, and Theorem 5.19 are constant-free. For instance, $\mathrm{EULER}^0_{\mathrm{add}} \in \mathrm{BP}((\mathrm{FP}^0_{\mathrm{add}})^{\#\mathrm{P}^0_{\mathrm{add}}}) = \mathsf{FP}^{\#\mathsf{P}}$ according to Remark 4.12. For the hardness note that

$$(\mathrm{FP}^0_{\mathrm{add}})^{\#\mathrm{P}^0_{\mathrm{add}}} = (\mathrm{FP}^0_{\mathrm{add}})^{\#\mathrm{CSAT}^0_{\mathrm{add}}} = (\mathrm{FP}^0_{\mathrm{add}})^{\mathrm{EULER}^0_{\mathrm{add}}}.$$

By taking Boolean parts und using Remark 4.12, we conclude $\mathsf{FP}^{\#\mathsf{P}} = \mathsf{FP}^{\mathrm{EULER}^0_{\mathrm{add}}}$.

The statement about $\mathrm{BETTI}^0_{\mathrm{add}}(k)$ is proved similarly. □

6. Summary. To facilitate the orientation of the reader, we have summarized the results of this paper in Figure 6.1. There an arrow denotes an inclusion, problems in square brackets are Turing-complete for the class at their left, and problems in curly brackets are many-one-complete for that class. The problems appearing in the figure are defined in the list below. We note that the completeness of SAT, $\mathrm{CSAT_{add}}$, $\mathrm{TSP}_{\mathbb{R}}$, $\mathrm{PM}_{\mathbb{R}}$, QBF, and DTRAO was already known.

SAT (*satisfiability*). Given a propositional formula φ, decide whether there is an assignment of Boolean values for the variables satisfying φ.

#SAT (*counting satisfiability*). Given φ as in SAT, count the number of satisfying assignments.

QBF (*quantified Boolean formulas*). Given a quantified Boolean formula, decide whether it is a tautology.

DTRAO (*digital theory of the reals with addition and order*). Given a sentence in the theory of the reals with addition and order, all of whose variables satisfy a constraint of the form $x = 0 \vee x = 1$, decide whether it is a tautology.

$\mathrm{TSP}_{\mathbb{R}}$ (*traveling salesman*). Given a complete graph G with real weights on the edges and $w \in \mathbb{R}$, decide whether there is a Hamilton circuit in G with weight at most w. (The weight of a subgraph is the sum of the weights of its edges.)

$\#\mathrm{TSP}_{\mathbb{R}}$ (*counting traveling salesman*). Given G and $w \in \mathbb{R}$ as in $\mathrm{TSP}_{\mathbb{R}}$, count the number of Hamilton circuits in G with weight at most w.

$\mathrm{PM}_{\mathbb{R}}$ (*weighted perfect matching*). Given a bipartite graph G with real weights on the edges and $w \in \mathbb{R}$, decide whether there is a perfect matching in G with weight at most w.

$\#\mathrm{PM}_{\mathbb{R}}$ (*counting weighted perfect matchings*). Given G and $w \in \mathbb{R}$ as in $\mathrm{PM}_{\mathbb{R}}$, count the number of perfect matchings in G with weight at most w.

$\mathrm{CSAT}_{\mathrm{add}}$ (*circuit satisfiability*). Decide whether the semilinear set given by an additive circuit is nonempty.

$\mathrm{DIM}_{\mathrm{add}}(d)$ (*dimension*). Given an additive circuit and $d \in \mathbb{N}$, decide whether the dimension of the semilinear set defined by the circuit is at least d.

$\mathrm{REACH}_{\mathrm{add}}$ (*reachability*). Given an additive circuit defining a semilinear set X and two points $s, t \in X$, decide whether s and t are in the same connected component of X.

$\#\mathrm{CSAT}_{\mathrm{add}}$ (*point counting*). Given an additive circuit defining a semilinear set X, compute the number of points in X.

$\#^{\mathrm{cc}}\mathrm{CSAT}_{\mathrm{add}}$ (*counting connected components*). Given an additive circuit defining a semilinear set X, compute the number of connected components of X.

$\mathrm{EULER}_{\mathrm{add}}$ (*Euler characteristic*). Given an additive circuit defining a closed semilinear set X, decide whether X is empty and, if not, compute its Euler characteristic.

$\mathrm{BETTI}_{\mathrm{add}}(k)$ (*Betti numbers*). Given an additive circuit defining a closed semilinear set X, compute the kth Betti number of X.

7. Open questions. We present some selected open problems.

PROBLEM 7.1. In this paper, we prove completeness with respect to Turing reductions. Do we also have completeness with respect to parsimonious reductions? For instance, how about the completeness of $\#\mathrm{TSP}_{\mathbb{R}}$ in $\mathrm{D}\#\mathrm{P}_{\mathrm{add}}$?

PROBLEM 7.2. What is the complexity to deciding connectedness of a semilinear set given by an additive circuit?

PROBLEM 7.3. In this paper we proved that computing the torsion-free part of the homology of semilinear sets is $\mathrm{FPAR}_{\mathrm{add}}$-complete. Is the complexity of computing the torsion part of this homology also $\mathrm{FPAR}_{\mathrm{add}}$-complete?

Acknowledgments. We thank Eric Allender and Saugata Basu for helpful discussions. We are grateful to the referees for many valuable comments, which helped to improve the presentation significantly.

REFERENCES

[1] J. BALCÁZAR, A. LOZANO, AND J. TORÁN, *The complexity of algorithmic problem on succinct instances*, in Computer Science: Research and Applications, Plenum Press, New York, 1992, pp. 351–377.

[2] J. L. BALCÁZAR, J. DÍAZ, AND J. GABARRÓ, *Structural Complexity* I, Springer-Verlag, Berlin, 1988.

[3] M. BEN-OR, *Lower bounds for algebraic computation trees*, in Proceedings of the 15th Annual ACM Symposium on Theory of Computing, Boston, MA, 1983, pp. 80–86.

[4] L. BLUM, F. CUCKER, M. SHUB, AND S. SMALE, *Complexity and Real Computation*, Springer-Verlag, New York, 1998.

[5] L. BLUM, M. SHUB, AND S. SMALE, *On a theory of computation and complexity over the real numbers*, Bull. Amer. Math. Soc. (N.S.), 21 (1989), pp. 1–46.

[6] A. BORODIN, *On relating time and space to size and depth*, SIAM J. Comput., 6 (1977), pp. 733–744.

[7] P. BÜRGISSER, *Completeness and Reduction in Algebraic Complexity Theory*, Algorithms Comput. Math. 7, Springer-Verlag, Berlin, 2000.

[8] P. BÜRGISSER, *Cook's versus Valiant's hypothesis*, Theoret. Comput. Sci., 235 (2000), pp. 71–88.

[9] P. Bürgisser, *Lower bounds and real algebraic geometry*, in Algorithmic and Quantitative Real Algebraic Geometry, S. Basu and L. Gonzales-Vega, eds., DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 60, AMS, Providence, RI, 2003, pp. 35–54.
[10] P. Bürgisser, M. Clausen, and M. Shokrollahi, *Algebraic Complexity Theory*, Grundlehren Math. Wiss. 315, Springer-Verlag, Berlin, 1997.
[11] P. Bürgisser and F. Cucker, *Counting Complexity Classes for Numeric Computations* II: *Algebraic and Semialgebraic Sets*, manuscript.
[12] F. Cucker and D. Grigoriev, *On the power of real Turing machines over binary inputs*, SIAM J. Comput., 26 (1997), pp. 243–254.
[13] F. Cucker, M. Karpinski, P. Koiran, T. Lickteig, and K. Werther, *On real Turing machines that toss coins*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 335–342.
[14] F. Cucker and P. Koiran, *Computing over the reals with addition and order: Higher complexity classes*, J. Complexity, 11 (1995), pp. 358–376.
[15] F. Cucker and M. Matamala, *On digital nondeterminism*, Math. Systems Theory, 29 (1996), pp. 635–647.
[16] H. Fournier and P. Koiran, *Are lower bounds easier over the reals?*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 507–513.
[17] H. Fournier and P. Koiran, *Lower bounds are not easier over the reals: Inside* PH, in Proceedings of the 27th International Colloquium on Automata, Languages, and Programming, Geneva, Switzerland, 2000, Lecture Notes in Comput. Sci. 1853, Springer-Verlag, Berlin, 2000, pp. 832–843.
[18] H. Galperin and A. Wigderson, *Succinct representation of graphs*, Inform. and Control, 56 (1983), pp. 183–198.
[19] J. v. z. Gathen, *Parallel arithmetic computations: A survey*, in Proceedings of the 12th Annual Symposium on Mathematical Foundations of Computer Science, Bratislava, Czechoslovakia, 1986, Lecture Notes in Comput. Sci. 233, J. Gruska, B. Rovan, and J. Wiedermann, eds., Springer-Verlag, Berlin, 1986, pp. 93–112.
[20] A. Hatcher, *Algebraic Topology*, Cambridge University Press, Cambridge, UK, 2002.
[21] N. Karmarkar, *A new polynomial time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.
[22] L. Khachijan, *A polynomial algorithm in linear programming*, Dokl. Akad. Nauk SSSR, 244 (1979), pp. 1093–1096 (in Russian); Soviet Math. Dokl., 20 (1979), pp. 191–194 (in English).
[23] K.-I. Ko, *Complexity of Real Functions*, Birkhäuser Boston, Boston, 1991.
[24] P. Koiran, *Computing over the reals with addition and order*, Theoret. Comput. Sci., 133 (1994), pp. 35–47.
[25] P. Koiran, *A weak version of the Blum, Shub, and Smale model*, J. Comput. System Sci., 54 (1997), pp. 177–189.
[26] H. Lewis and C. Papadimitriou, *Symmetric space-bounded computation*, Theoret. Comput. Sci., 19 (1982), pp. 161–187.
[27] K. Meer, *Counting problems over the reals*, Theoret. Comput. Sci., 242 (2000), pp. 41–58.
[28] S. Meiser, *Point location in arrangements of hyperplanes*, Inform. and Comput., 106 (1993), pp. 286–303.
[29] F. Meyer auf der Heide, *A polynomial linear search algorithm for the n-dimensional knapsack problem*, J. Assoc. Comput. Mach., 31 (1984), pp. 668–676.
[30] F. Meyer auf der Heide, *Fast algorithms for n-dimensional restrictions of hard problems*, J. Assoc. Comput. Mach., 35 (1988), pp. 740–747.
[31] C. Michaux, *Une remarque à propos des machines sur* $\mathbb{R}$ *introduites par Blum, Shub et Smale*, C. R. Acad. Sci. Paris Sér I Math., 309 (1989), pp. 435–437.
[32] K. Mulmuley, *A fast parallel algorithm to compute the rank of a matrix over an arbitrary field*, Combinatorica, 7 (1987), pp. 101–104.
[33] J. R. Munkres, *Elements of Algebraic Topology*, Addison-Wesley, Menlo Park, CA, 1984.
[34] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
[35] J. Reif, *Complexity of the mover's problem and generalizations*, in Proceedings of the 20th Annual Symposium on Foundations of Computer Science, San Juan, PR, 1979, pp. 421–427.
[36] J. Reif, *Complexity of the generalized mover's problem*, in Planning, Geometry and Complexity of Robot Motion, J. Schwartz, M. Sharir, and J. Hopcroft, eds., Ablex Publishing Corporation, Norwood, NJ, 1987, pp. 267–281.
[37] E. Tardos, *A strongly polynomial algorithm to solve combinatorial linear programs*, Oper. Res., 34 (1986), pp. 250–256.
[38] S. Toda, PP *is as hard as the polynomial-time hierarchy*, SIAM J. Comput., 20 (1991), pp. 865–

877.

[39] S. Toda and O. Watanabe, *Polynomial time* 1*-Turing reductions from* #PH *to* #P, Theoret. Comput. Sci., 100 (1992), pp. 205–221.

[40] L. Valiant, *Completeness classes in algebra*, in Proceedings of the 11th Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1979, pp. 249–261.

[41] L. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.

[42] L. G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput., 8 (1979), pp. 410–421.

[43] L. Valiant, *Reducibility by algebraic projections*, in Logic and Algorithmic: An International Symposium Held in Honor of Ernst Specker, Monograph. Enseign. Math. 30, 1982, pp. 365–380.

[44] A. Yao, *Algebraic decision trees and Euler characteristic*, in Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992, pp. 268–277.

[45] A. Yao, *Decision tree complexity and Betti numbers*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, Montréal, QC, Canada, 1994, pp. 615–624.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 261–268

AN EFFICIENT POLYNOMIAL TIME APPROXIMATION SCHEME FOR THE CONSTRAINED MINIMUM SPANNING TREE PROBLEM USING MATROID INTERSECTION*

REFAEL HASSIN† AND ASAF LEVIN†

Abstract. Given an undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, nonnegative integers c_e and d_e for each edge $e \in E$, and a bound D, the constrained minimum spanning tree problem (CST) is to find a spanning tree $T = (V, E_T)$ such that $\sum_{e \in E_T} d_e \leq D$ and $\sum_{e \in E_T} c_e$ is minimized. We present an efficient polynomial time approximation scheme (EPTAS) for this problem. Specifically, for every $\epsilon > 0$ we present a $(1 + \epsilon)$-approximation algorithm with time complexity $O((\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})} n^4)$. Our method is based on Lagrangian relaxation and matroid intersection.

Key words. approximation algorithm, spanning tree, matroid intersection, bicriteria optimization

AMS subject classifications. 90C27, 90C59

DOI. 10.1137/S0097539703426775

1. Introduction.

Given an undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, nonnegative integers c_e and d_e for each edge $e \in E$, and a bound D, the constrained minimum spanning tree problem (CST) is to find a spanning tree $T = (V, E_T)$ such that $\sum_{e \in E_T} d_e \leq D$ and $\sum_{e \in E_T} c_e$ is minimized. We refer to c_e and d_e of an edge e as its *cost* and *length*, respectively. Thus the problem we consider is that of finding a minimum cost spanning tree whose total length is at most D.

Aggarwal, Aneja, and Nair [1] proved that the problem is weakly NP-hard. With respect to approximation algorithms, Goemans and Ravi [9] presented a polynomial time approximation scheme (PTAS), i.e., a $(1 + \epsilon)$-approximation algorithm for every $\epsilon > 0$, with time complexity $O(n^{O(\frac{1}{\epsilon})} \log C_{max})$, where C_{max} is the largest cost. Their algorithm is based on Lagrangian relaxation (for an earlier work on this problem that uses a Lagrangian relaxation, see Jornsten and Migdalas [13]) and uses the adjacency relations for matroids. Recently, Hong, Chung, and Park [12] showed a pseudopolynomial time algorithm for this problem that is based on a two-variable extension of the tree-matrix theorem. We note that developing a fully polynomial time approximation scheme based on their pseudopolynomial time algorithm is not straightforward. Instead, [12] contains a fully polynomial *bicriteria* approximation scheme. Currently, the question whether there exists a fully polynomial time approximation scheme to CST is open. Andersen, Jornsten, and Lind [2] presented heuristic algorithms for CST.

In this paper we provide the first efficient polynomial time approximation scheme (EPTAS) to this problem: for every $\epsilon > 0$ we provide a $(1 + \epsilon)$-approximation with time complexity $g(\frac{1}{\epsilon})poly(n)$, where *poly* is some polynomial function, and g is any function. Specifically, the complexity of our algorithm is $O((\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})} n^4)$. Our method adopts the basic ideas of Goemans and Ravi [9] and adds to them a novel application of a matroid intersection algorithm. We note that our method (as well as Goemans

*Received by the editors April 30, 2003; accepted for publication (in revised form) July 11, 2003; published electronically January 22, 2004.
http://www.siam.org/journals/sicomp/33-2/42677.html
†Department of Statistics and Operations Research, Tel-Aviv University, Tel-Aviv 69978, Israel (hassin@post.tau.ac.il, levinas@post.tau.ac.il).

and Ravi's) provides a similar EPTAS using the same analysis to finding a minimum cost base of a matroid with total length at most D, as long as there is a polynomial time independence oracle for the matroid (for example, a union of k forests).

Lagrangian relaxation is a well-known method to approximate a constrained optimization problem.

We now present some basic definitions and results in matroid theory (see [14] or [7]). An ordered pair $M = (E, \mathcal{F})$, where E is a set of elements and $\mathcal{F}$ is a family of subsets of E, is called a *matroid* if the following properties are satisfied: $\emptyset \in \mathcal{F}$, if $S \in \mathcal{F}$ and $S' \subseteq S$, then $S' \in \mathcal{F}$, and if $S_p, S_{p+1} \in \mathcal{F}$ are subsets with $p, p+1$ elements, respectively, then there is $e \in S_{p+1} \setminus S_p$ such that $S_p \cup \{e\} \in \mathcal{F}$. An element of $\mathcal{F}$ is an *independent set*. A maximal independent set is a *base* of the matroid. A *polynomial time independence oracle* for a matroid M is a polynomial time algorithm such that for a given subset $S \subseteq E$ the algorithm checks whether or not $S \in \mathcal{F}$. Given a matroid M with an element cost function c, the greedy algorithm finds a minimum (total) cost base of M. Therefore, for a matroid with a polynomial time independence oracle, the problem of finding a minimum cost base is polynomially solvable. Given a pair of matroids $M = (E, \mathcal{F})$ and $M' = (E, \mathcal{F}')$ with polynomial time independence oracles over common element set E and element cost function c, there is a polynomial time algorithm that computes a minimum cost common base of M and M' (i.e., a minimum cost subset S such that S is a base of both M and M') [3]. Computing the minimum total common base of two matroids is one of the well-studied variants of matroid intersection.

Matroid intersection is one of the strongest available tools that one can use in polynomial time algorithms. However, we are aware of only a few uses of it in the design of approximation algorithms (see, for example, Chalasani and Motwani [5]). Therefore, we also contribute to extending the uses of matroid intersection.

Denote by OPT the optimum value for our problem. Given a spanning tree $T = (V, E_T)$ and a function f defined over E, we denote $f(T) = \sum_{e \in E_T} f(e)$.

Goemans and Ravi presented the following results:

1. Given an $(\alpha, 1)$-approximation algorithm with time complexity T_{alg} (i.e., an algorithm that provides output T such that $d(T) \leq \alpha D$ and $c(T) \leq OPT$), there is a $(1+\epsilon)\alpha$-approximation algorithm with time complexity $O(T_{alg} \log_{1+\epsilon} C_{max})$ where C_{max} is the largest cost. To prove this, they suggested applying a binary search over OPT and replacing the roles of c and d in the bicriteria approximation algorithm.
2. A $(2, 1)$-approximation algorithm with time complexity $O(m \log^2 n + n \log^3 n)$; with the former result this gives a $2(1+\epsilon)$-approximation algorithm with time complexity $O((m \log^2 n + n \log^3 n) \log_{1+\epsilon} C_{max})$.
3. A $(1+\epsilon, 1)$-approximation algorithm with time complexity $O(n^{O(\frac{1}{\epsilon})})$ for every $1 > \epsilon > 0$; with the former result this gives a $(1+\epsilon)$-approximation algorithm with time complexity $O(n^{O(\frac{1}{\epsilon})} \log_{1+\epsilon} C_{max})$ for every $\epsilon > 0$.

REMARK 1. *The time complexity in result* 1 *can be reduced to* $O(T_{alg} \log \log_{1+\epsilon} n + T_{MST} \log n)$, *where* T_{MST} *is the time needed to compute a minimum spanning tree.*

Proof. We apply Hassin's method [11] and apply approximate geometric-mean binary search instead of arithmetic-mean binary search. This reduces the number of iterations to $\log_2 \log_{1+\epsilon} \frac{UB}{LB}$, where UB and LB are upper and lower bounds on OPT, respectively.

We use the method of Lorenz and Raz [15] to obtain good bounds. Let $S_c = \{c_e\}_{e \in E}$. Given a threshold value $c \in S_c$, we consider the subgraph $G_c = (V, \{e \in E : c_e \leq c\})$. We find a minimum value $c \in S_c$ such that G_c contains a spanning tree T

such that $d(T) \le D$, and then $c \le OPT \le (n-1)c$. Thus, by applying binary search with $O(\log n)$ iterations on S_c, we obtain bounds LB and UB such that $\frac{UB}{LB} \le n-1$. The time complexity of the binary search is $O(T_{MST} \log n)$. □

Assume that $\epsilon \le \frac{1}{4}$. We first compute a $2(1+\epsilon)$-approximation solution using result 2. The complexity of this phase is $O((m + n\log n)\log^2 n \log\log_{1+\epsilon} n)$. Denote by $\hat{C}$ the cost of this solution. Then $OPT \le \hat{C} \le 2(1+\epsilon)OPT$. We remove from E the edge set $\{e \in E : c_e > \hat{C}\}$.

2. The algorithm. First, we present the main tools that enable us to improve Goemans and Ravi's result. Given $\epsilon > 0$ and $\hat{C}$, we partition E into the following edge sets: $E_0 = \{e \in E : c_e < \epsilon\hat{C}\}$, and for $1 \le i \le I = \lceil \frac{1-\epsilon}{\epsilon^2} \rceil$, $E_i = \{e \in E : (\epsilon + (i-1)\epsilon^2)\hat{C} \le c_e < (\epsilon + i\epsilon^2)\hat{C}\}$.

Our algorithm enumerates all the possible vectors with $I+1$ nonnegative integer entries $(n_0, n_1, \ldots, n_I)$ such that the following conditions hold:

1. $\sum_{i=0}^{I} n_i = n-1$.
2. $\sum_{i=1}^{I} n_i \le \frac{1}{\epsilon}$.

For each such vector, our algorithm finds an approximate solution with exactly n_i edges from the set E_i if such a solution exists. Note that an optimal solution for CST may have at most $\frac{1}{\epsilon}$ edges with cost at least ϵOPT and, therefore, the second condition holds for every optimal solution for CST.

LEMMA 2. *There are $O(I^{\frac{1}{\epsilon}})$ vectors that satisfy both conditions.*

Proof. We have at most $\frac{1}{\epsilon}$ items in the last I components. We consider a bucket for each set E_i, $i \ge 1$, and we have another bucket $i = 0$ that collects $\frac{1}{\epsilon} - \sum_{i=1}^{I} n_i$ items. We have to place $\frac{1}{\epsilon}$ items in the $I+1$ buckets. Therefore, the number of possibilities is at most $(I+1)^{\frac{1}{\epsilon}} = O(I^{\frac{1}{\epsilon}})$. □

Given a cost function $c^{\lambda}(e)$, $e \in E$, and a vector $(n_0, n_1, \ldots, n_I)$, we can find a minimum cost spanning tree with n_i edges from the set E_i (if such a spanning tree exists). We do it via matroid intersection in the following manner: we define a *partition matroid* over the ground set E such that a set $E' \subseteq E$ is independent in this matroid if and only if $|E' \cap E_i| \le n_i$ for $i = 0, 1, \ldots, I$. We define a second matroid over E, the *graphic matroid*, where a set $E' \subseteq E$ is independent if and only if the graph (V, E') does not contain a cycle. We find a minimum cost common base of these two matroids. This is done by computing a minimum cost set (if one exists) that is independent in both matroids and has cardinality $n-1$. Since the number of sets in the partition matroid is $O(\frac{1}{\epsilon})$, we can find a minimum cost basis in the intersection of the two matroids in $O(mn + \frac{n^2}{\epsilon})$ time (see Brezovec, Cornuejols, and Glover [4]).

Next we describe the use of Lagrangian relaxation for our problem (see [9] and [13] for a similar idea). Given a graph $G = (V, E)$, a value $\hat{C}$, and $(n_0, n_1, \ldots, n_I)$, let S denote the set of incidence vectors of common bases of the graphic matroid and the partition matroid. Our problem is

$$
\begin{aligned}
C = \min \sum_{e \in E} c_e x_e & \\
\text{subject to} \quad & x \in S \\
& \sum_{e \in E} d_e x_e \le D.
\end{aligned} \tag{1}
$$

Consider the budget constraint (1) as the complicating constraint. We can obtain a lower bound on the optimum value C by dualizing it, and for any $\lambda \ge 0$ we consider

the problem defined as

$$C(\lambda) = \min \sum_{e \in E} (c_e + \lambda d_e) x_e - \lambda D$$
$$\text{subject to} \quad x \in S.$$

We let $c^\lambda(e) = c_e + \lambda d_e$ for every edge $e \in E$, and note that the problem $C(\lambda)$ is, exactly, to compute the minimum cost basis in the intersection of the two matroids as described above. In order to find the best lower bound on C we maximize $C(\lambda)$ over all $\lambda \geq 0$ to obtain

$$LR = \max_{\lambda \geq 0} C(\lambda).$$

It is well known that since $C(\lambda)$ is a minimum of a set of linear functions (one for every $x \in S$), $C(\lambda)$ is concave and piecewise linear (see, for example, Fact 1.1 in [10]). Let λ^* denote a value of λ that maximizes $C(\lambda)$, and let $c^*(e) = c^{\lambda^*}(e)$ for every $e \in E$.

To solve the Lagrangian relaxation, we need to find the minimum base in the matroid intersection with respect to c^* without knowing λ^*. In order to do so, we apply the combinatorial algorithm in [4] (the dual algorithm in section 3 of [4]) that solves the minimum cost basis in the intersection of the two matroid problem where one of them is a partition matroid. In each comparison we need to compare $a_1\lambda^* + b_1$ and $a_2\lambda^* + b_2$. This can be done by computing the breakpoint $\lambda_{br} = \frac{b_2 - b_1}{a_1 - a_2}$ (if $a_1 \neq a_2$—otherwise the comparison is independent of λ^*), and then we need to determine whether $\lambda^* \leq \lambda_{br}$, $\lambda^* = \lambda_{br}$, or $\lambda^* \geq \lambda_{br}$. This can be done using a pair of matroid intersection computations (with $\lambda = \lambda_{br}^-$ and with $\lambda = \lambda_{br}^+$). This leads to an $O([mn + \frac{n^2}{\epsilon}]^2) = O(m^2n^2 + \frac{n^4}{\epsilon^2})$ algorithm that solves (optimally) the Lagrangian relaxation.

Our algorithm is based on solving the Lagrangian relaxation, as well as deriving a good spanning tree out of it. We use an adjacency relationship on the polytope defined by $conv(S)$, where S is the set of incidence vectors of the common bases in the intersection of two matroids. This adjacency relationship holds when one of the matroids is a partition matroid (the other matroid can vary). The next lemma generalizes Lemma 3.2 in [9] (where it was stated for the trivial partition matroid, the case $I = 0$), and Lemma 9 in [8] (for a partition matroid with two sets, $I = 1$).

LEMMA 3. *Let $n_0, n_1, \ldots, n_I$ be the number of edges from $E_0, E_1, \ldots, E_I$, respectively, in an optimal solution. Let $\mathcal{O}$ denote the set of minimum cost common bases in the intersection of the two matroids with respect to c^*, when the partition matroid is defined with $n_0, n_1, \ldots, n_I$. Let $T, T' \in \mathcal{O}$. Then there is a series of trees $T = T_0 = (V, E_{T_0}), T_1 = (V, E_{T_1}), \ldots, T' = T_l = (V, E_{T_l})$ that satisfies the following:*

1. *$T_j \in \mathcal{O}$, for $j = 0, \ldots, l$.*
2. *Let $E^j = T_j \setminus T_{j+1}$ and $E'^j = T_{j+1} \setminus T_j$. Then $|E^j \cap E_i| = |E'^j \cap E_i| \leq 1$ for $i = 0, \ldots, I$ and $j = 0, \ldots, l-1$.*

Proof. We construct a path from T to T' of edge swaps. We prove the lemma by induction over $|T' \setminus T|$. For $|T' \setminus T| = 1$ the lemma holds trivially. Assume the lemma holds for any pair of optimal trees T, T' such that $|T' \setminus T| = x - 1$; we prove it for $|T' \setminus T| = x$.

We consider a perturbed cost function c' (for a small enough value of ϵ') defined as follows:

$$c'(e) = \begin{cases} \infty & \text{if } e \notin T \cup T', \\ c^*(e) + \epsilon' & \text{if } e \in T \setminus T', \\ c^*(e) - \epsilon' & \text{if } e \in T' \setminus T, \\ c^*(e) & \text{if } e \in T \cap T'. \end{cases}$$

We take T to be the initial solution to the primal algorithm in [4]. T is not optimal with respect to c' (because T' is the unique optimal solution with respect to c'), and therefore the algorithm finds a negative cycle in a graph S. We first define an auxiliary bipartite directed graph $B = (V, V', A)$ as follows (see [3] and [4]): For every $e \in T$ we have a node in V, and for every $e' \in E \setminus T$ we have a node in V'. For every $e_i \in T$ and $e'_j \in E \setminus T$ we construct an arc (e_i, e'_j) with cost $c(e_i, e'_j) = c'(e_i) - c'(e_j)$ provided that $(T \setminus \{e_i\}) \cup \{e'_j\}$ is independent in the graphic matroid, and we construct an arc (e'_j, e_i) with cost $c(e'_j, e_i) = 0$ provided that $(T \setminus \{e_i\}) \cup \{e'_j\}$ is independent in the partition matroid (i.e., e_i and e'_j belong to the same set of the partition). Since for $e \notin T \cup T'$, $c'(e) = \infty$, a negative cycle in B does not contain e. S is obtained from B (see [4]) by identifying the set of nodes corresponding to elements from $(E \setminus T) \cap E_i$ in V' to a single vertex v_i for every i, and removing parallel edges.

By Theorem 4 in [4], there is a common base T'' that has a smaller cost than T with respect to c' (and therefore, $T'' \in \mathcal{O}$) such that T'' is obtained from T by swapping edges along a negative cycle C that has a minimum number of arcs (among all the negative cycles in S). Suppose that the nodes along the cycle C correspond to edges $(e_1, e'_1, e_2, e'_2, \ldots, e_l, e'_l, e_1)$ such that $e_k \in E_{i_k}$ for $k = 1, \ldots, l$. C is simple, and therefore, $i_k \neq i_{k'}$ $\forall k \neq k'$ and $T'' \in \mathcal{O}$.

$T'' \subseteq T \cup T'$ and $T'' \neq T$, and therefore, $|T'' \setminus T'| < |T \setminus T'| = x$. Therefore, by the induction assumption, the lemma's claim holds. □

Our main result will follow from the next theorem.

THEOREM 4. *Let $n_0, n_1, \ldots, n_I$ be the number of edges from $E_0, E_1, \ldots, E_I$, respectively, in an optimal solution for CST. Let $\mathcal{O}$ denote the set of minimum cost common bases in the intersection of the two matroids with respect to c^*, when the partition matroid is defined with $n_0, n_1, \ldots, n_I$. Then there exists $T = (V, E_T) \in \mathcal{O}$ such that*

$$c(T) \leq LR + \sum_{i:n_i>0} (\max_{e \in E_i}\{c_e\} - \min_{e \in E_i}\{c_e\})$$

and

$$d(T) \leq D.$$

Proof. Let $\lambda = \lambda^* - \epsilon'$, where $\epsilon' > 0$. There is $T^- \in \mathcal{O}$ that is optimal also with respect to c^λ (for a small enough value of ϵ'). Since $c^\lambda(T^-) = C(\lambda) \leq C(\lambda^*)$, we obtain

$$c(T^-) + (\lambda^* - \epsilon')[d(T^-) - D] \leq c(T^-) + \lambda^*[d(T^-) - D] = LR.$$

Therefore, $d(T^-) \geq D$, and thus $c(T^-) \leq LR$. Similarly, for $\lambda = \lambda^* + \epsilon'$, there is $T^+ \in \mathcal{O}$ such that $d(T^+) \leq D$ and $c(T^+) \geq LR$.

By Lemma 3, there is a series $T^- = T_0, T_1, \ldots, T_l = T^+$ of spanning trees that satisfies the properties of the lemma. Since $c(T^-) \leq LR$ and $c(T^+) \geq LR$, there is an index j such that $c(T_j) \leq LR$ and $c(T_{j+1}) \geq LR$ (and therefore, $d(T_{j+1}) \leq D$). Therefore,

$$c(T_{j+1}) \leq c(T_j) + \sum_{i:n_i>0} (\max_{e \in E_i}\{c_e\} - \min_{e \in E_i}\{c_e\}). \qquad \square$$

REMARK 5.

$$\sum_{i:n_i>0} (\max_{e\in E_i}\{c_e\} - \min_{e\in E_i}\{c_e\}) \le 4(1+\epsilon)\epsilon OPT \le 5\epsilon OPT.$$

Proof. Note that $\max_{e\in E_0}\{c_e\} \le \epsilon\hat{C} \le 2(1+\epsilon)\epsilon OPT$, $\min_{e\in E_0}\{c_e\} \ge 0$, and therefore, the term that corresponds to $i = 0$ is at most $2(1+\epsilon)\epsilon OPT$. There are at most $\frac{1}{\epsilon}$ other terms in the summation. By the definition of E_i, each of them contributes at most $\epsilon^2\hat{C} \le 2(1+\epsilon)\epsilon^2 OPT$, and together they contribute at most $2(1+\epsilon)\epsilon OPT$. The second inequality follows for $\epsilon \le \frac{1}{4}$. ∎

REMARK 6. *The proof of Lemma 3 leads naively to an $O(\frac{n^3}{\epsilon})$ algorithm that, given T^- and T^+, finds T that satisfies the claimed properties.*

Proof. We construct the path of trees as in Lemma 3. In each step we first construct the graph S. S contains $O(n)$ vertices and $O(\frac{n}{\epsilon})$ edges. We construct S in $O(\frac{n^2}{\epsilon})$ time by identifying for each e_i the set $\{e'_j \in T' : (T\setminus\{e_i\})\cup\{e'_j\}$ is independent in the graphic matroid$\}$ in $O(n)$ time. Next, we find a negative cycle with a minimum number of edges using the Bellman–Ford algorithm in $O(\frac{n^2}{\epsilon})$ time. Finally, we identify the next tree along the path in $O(n)$ time. The total time complexity of a single step is $O(\frac{n^2}{\epsilon})$ time. Since the total number of trees along this path is $O(n)$, we conclude that computing the entire path takes $O(\frac{n^3}{\epsilon})$ time. ∎

Theorem 4 and Remarks 5 and 6 motivate the following algorithm: First, using the $2(1+\epsilon)$-approximation algorithm in [9], find a $2(1+\epsilon)$-approximation value $\hat{C}$ of OPT ($OPT \le \hat{C} \le 2(1+\epsilon)OPT$). Second, enumerate all the possible vectors $(n_0, n_1, \ldots, n_I)$, and for each of them apply the following procedure: Compute λ^* that optimizes the Lagrangian relaxation. Among all the optimum common bases (in the matroid intersection) for the cost function c^*, find a base that satisfies the conditions of Theorem 4.

Finally, pick the tree that minimizes $c(T)$ (among all the possible vectors $(n_0, n_1, \ldots, n_I)$ that lead to a feasible tree).

We note that for the correct guess of the vector $(n_0, n_1, \ldots, n_I)$, the cost of the minimum cost base is exactly OPT. Since $\hat{C} \le 2(1+\epsilon)OPT$, the tree that we find costs (by Theorem 4 and Remark 5) at most $OPT + 5\epsilon OPT = (1+5\epsilon)OPT$, and its length is at most D.

Computing $\hat{C}$ takes $O((m + n\log n)\log^2 n \log\log_{1+\epsilon} n)$. By Lemma 2, there are $O((\frac{1}{\epsilon^2})^{\frac{1}{\epsilon}})$ distinct vectors $(n_0, n_1, \ldots, n_I)$. For each of them we solve the Lagrangian relaxation in $O(m^2n^2 + \frac{n^4}{\epsilon^2})$ time, and by Remark 6 we find T in another $O(\frac{n^3}{\epsilon})$ time. Therefore, the total complexity of the algorithm is $O((\frac{1}{\epsilon^2})^{\frac{1}{\epsilon}}(m^2n^2))$.

We conclude with the following theorem.

THEOREM 7. *For every $\epsilon > 0$, there is a $(1+\epsilon)$-approximation algorithm that runs in $O(O(\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}(m^2n^2))$ time.*

3. Improved complexity. We first note that if there is a parallel algorithm for the minimum cost basis in the intersection of the two matroid problem, then it may be combined (similar to the use in [9]) with Megiddo's method [16] to improve the time complexity for solving the Lagrangian relaxation. However, we are not aware of such an algorithm.

In this section we provide a faster algorithm that solves the Lagrangian relaxation. Our algorithm solves the Lagrangian relaxation in $O(\frac{n^4}{\epsilon^2})$ time.

To present the algorithm we note that for every $\lambda \ge 0$ and $i \ge 0$, an edge $e \in E_i$

that participates in the minimum cost common base of the two matroids belongs to a minimum cost (with respect to $c^\lambda = c + \lambda d$) spanning forest $F_i(\lambda)$ of (V, E_i). $c^\lambda(F_i(\lambda))$ is a piecewise linear function of λ, and in each breakpoint of it $F_i(\lambda)$ changes by an edge swap (or a set of edge swaps in degenerate cases). Therefore, in each breakpoint there is a pair of edges that have the same cost. We define a set of potential breakpoints as $\{\lambda : \exists e, e' \in E,\ c^\lambda(e) = c^\lambda(e')\}$. Then, every breakpoint of $c^\lambda(F_i(\lambda))$ for all i belongs to the set of potential breakpoints. There are $O(m^2)$ potential breakpoints.

In a preprocessing step we compute the set of breakpoints. Afterwards, we apply a binary search over this set. Therefore, we find an interval (λ_1, λ_2) that contains λ^*, and such that no $F_i(\lambda)$ has a breakpoint in this interval beside λ^*.[1] Then, we compute $F_i(\hat{\lambda})$ for every i such that $n_i > 0$ for some $\hat{\lambda} \in (\lambda_1, \lambda_2)$. Now we remove from the two matroids all the edges that do not participate in $\bigcup_{i:n_i>0} F_i(\hat{\lambda})$. In the resulting common ground set, the number of elements is $O(\frac{n}{\epsilon})$. The total complexity of this preprocessing step is $O(\frac{mn \log n}{\epsilon})$.

We now apply our previous algorithm for solving the Lagrangian relaxation in $O(m^2n^2 + \frac{n^4}{\epsilon^2})$ to the resulting common ground set. The complexity of this procedure is $O(\frac{n^4}{\epsilon^2})$. By using the algorithm of this section in our approximation algorithm, we obtain the following theorem.

THEOREM 8. *For every $\epsilon > 0$ there is a $(1+\epsilon)$-approximation algorithm with time complexity $O(O(\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}(n^4))$.*

Note added in proof. By using the matroid intersection algorithm of Frederickson and Srinivas [8] instead of the algorithm of Brezovec, Cornuejols, and Glover [4], the complexity of our algorithm reduces to $O(O(\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}n^3)$.

Acknowledgment. We are grateful to Arie Tamir for many useful comments and in particular for helping us improve the time complexity of our algorithm from that stated in Theorem 7 to that stated in Theorem 8.

REFERENCES

[1] V. AGGARWAL, Y. ANEJA, AND K. NAIR, *Minimal spanning tree subject to a side constraint*, Comput. Oper. Res., 9 (1982), pp. 287–296.

[2] K. A. ANDERSEN, K. JORNSTEN, AND M. LIND, *On bicriterion minimal spanning trees: An approximation*, Comput. Oper. Res., 23 (1996), pp. 1171–1182.

[3] C. BREZOVEC, G. CORNUEJOLS, AND F. GLOVER, *Two algorithms for weighted matroid intersection*, Math. Program., 36 (1986), pp. 39–53.

[4] C. BREZOVEC, G. CORNUEJOLS, AND F. GLOVER, *A matroid algorithm and its application to the efficient solution of two optimization problems on graphs*, Math. Program., 42 (1988), pp. 471–487.

[5] P. CHALASANI AND R. MOTWANI, *Approximating capacitated routing and delivery problems*, SIAM J. Comput., 28 (1999), pp. 2133–2149.

[6] R. COLE, *Slowing down sorting networks to obtain faster sorting algorithms*, J. ACM, 34 (1987), pp. 200–208.

[7] W. J. COOK, W. H. CUNNINGHAM, W. R. PULLEYBLANK, AND A. SCHRIJVER, *Combinatorial Optimization*, John Wiley and Sons, New York, 1998.

[8] G. N. FREDERICKSON AND M. A. SRINIVAS, *Algorithms and data structures for an expanded family of matroid intersection problems*, SIAM J. Comput., 18 (1989), pp. 112–138.

[1] By using Cole's sorting network [6] we can find the interval (λ_1, λ_2) using $O(\log n)$ calls to minimum cost common base of the two matroids and additional $O(m)$ time without explicitly computing the set of $O(m^2)$ potential breakpoints.

[9] M. X. Goemans and R. Ravi, *The constrained minimum spanning tree problem*, in Proceedings of SWAT 96, Lecture Notes in Comput. Sci. 1097, Springer-Verlag, New York, 1996, pp. 66–75.

[10] D. Gusfield, *Parametric combinatorial computing and a problem of program module distribution*, J. ACM, 30 (1983), pp. 551–563.

[11] R. Hassin, *Approximation schemes for the restricted shortest path problem*, Math. Oper. Res., 17 (1992), pp. 36–42.

[12] S.-P. Hong, S.-J. Chung, and B. H. Park, *A fully polynomial bicriteria approximation scheme for the constrained spanning tree problem*, Oper. Res. Lett., to appear.

[13] K. Jornsten and S. Migdalas, *Designing a minimal spanning tree network subject to a budget constraint*, Optimization, 19 (1988), pp. 475–484.

[14] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[15] D. H. Lorenz and D. Raz, *A simple efficient approximation scheme for the restricted shortest path problem*, Oper. Res. Lett., 28 (2001), pp. 213–219.

[16] N. Megiddo, *Applying parallel computation algorithms in the design of serial algorithms*, J. ACM, 30 (1983), pp. 852–865.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 269–285

SHAPE FITTING WITH OUTLIERS*

SARIEL HAR-PELED† AND YUSU WANG‡

Abstract. Given a set $\mathcal{H}$ of n hyperplanes in $\mathbb{R}^d$, we present an algorithm that ε-approximates the extent between the top and bottom k levels of the arrangement of $\mathcal{H}$ in time $O(n+(k/\varepsilon)^c)$, where c is a constant depending on d. The algorithm relies on computing a subset of $\mathcal{H}$ of size $O(k/\varepsilon^{d-1})$, in near linear time, such that the k-level of the arrangement of the subset approximates that of the original arrangement. Using this algorithm, we propose efficient approximation algorithms for shape fitting with outliers for various shapes. These are the first algorithms to handle outliers efficiently for the shape fitting problems considered.

Key words. shape fitting, outliers, approximation

AMS subject classifications. 68U05, 65D99, 52B05, 52B10

DOI. 10.1137/S0097539703427963

1. Introduction.

Shape fitting, a fundamental problem in computational geometry, computer vision, machine learning, data mining, and many other areas, is concerned with finding the best shape fitting a given input. For example, a widely used shape fitting problem asks for a shape that best fits a set of points P under some "fitting" criterion. Choices of such shapes include points, lines, hyperplanes, and spheres. One typical criterion for measuring how well a shape γ fits P, denoted as $\mu(P,\gamma)$, is the maximum distance between a point of P and its nearest point on γ, i.e., $\mu(P,\gamma) = \max_{p\in P} \min_{q\in\gamma} d(p,q)$. One would then like to find the extent measure of P, defined as $\mu(P) = \min_\gamma \mu(P,\gamma)$, where the minimum is taken over a family of shapes. For example, the problem of finding the point (resp., line) that fits P best is the same as finding the minimum radius sphere (resp., cylinder) enclosing P, and the problem of finding the hyperplane (resp., sphere, cylinder) that fits P best is the same as finding the smallest width slab (resp., spherical shell, cylindrical shell) containing P.

The exact algorithms for shape fitting are generally expensive; e.g., the best-known algorithms for computing the smallest volume bounding box or tetrahedron containing P in $\mathbb{R}^3$ require $O(n^3)$ time . Consequently, attention has shifted to developing approximation algorithms for computing extent measures. Recently, Agarwal et al. [3, 4, 19] provided a general technique for approximate shape fitting in low dimensions. Their technique relied on using linearization to reduce the problem into a convex shape fitting problem and then solve it by known convex shape approximation techniques. The most striking features of this technique are its wide applicability to numerous shape fitting problems, and that the resulting running time is $O(n+1/\varepsilon^c)$, where n is the input size, c is a constant that depends on the problem at hand, and ε is the quality of approximation requested.

*Received by the editors May 14, 2003; accepted for publication (in revised form) September 10, 2003; published electronically January 22, 2004. A preliminary version of this paper appeared in *Proceedings of the 19th Annual ACM Symposium on Computational Geometry*, 2003, pp. 29–38.
http://www.siam.org/journals/sicomp/33-2/42796.html

†Department of Computer Science, DCL 2111, University of Illinois, 1304 West Springfield Ave., Urbana, IL 61801 (sariel@uiuc.edu). This author was supported by NSF CAREER award CR-0132901.

‡Department of Computer Science, Duke University, Durham, NC 27708-0129 (wys@cs.duke.edu). This author was supported by NSF grants ITR-333-1050 and CCR-02-04118.

However, in the real world, noise in the input is omnipresent and one has to assume that some of the input points are noise and as such should be ignored. To a certain extent, the problem of handling outliers in computational geometry is still open; see [7, 15, 22] for relevant results. While those results provide relatively efficient solutions, most of them are restricted to two and three dimensions (where intuitively the k-level has relatively low complexity) and are restricted in the type of problems that they can solve. For additional results about outliers in general, see [8, 14].

In this paper, we present approximation algorithms that can handle outliers efficiently in two or higher dimensions for a broad collection of shape fitting problems. For example, given a set P of n points in $\mathbb{R}^d$, we can find approximately the smallest cylinder that contains at least $n-k$ points of P in $O(n+(k/\varepsilon)^c)$ time, where c is a constant depending only on d.[1] Note that for a fixed ε, and a moderately small k, this is a linear time algorithm. Our algorithm can solve all the shape fitting problems considered in [3, 4, 19] efficiently under the additional constraint of k outliers. For most of those problems, this is the first efficient approximation algorithm to handle outliers.

Interestingly enough, our algorithm relies on computing a *coreset* to the given point set in *linear* time. Namely, we compute a subset $\mathfrak{C}$ of the points of cardinality $k/\varepsilon^{O(1)}$ such that instead of solving the problem on the original point set, one can solve it on $\mathfrak{C}$. In particular, in some cases, we just plug in the (exact) algorithms of [7, 12, 15, 22] on $\mathfrak{C}$ and get an efficient approximation algorithm.

To facilitate this, we investigate a related question, which is interesting in its own right: Given a set $\mathcal{H}$ of n hyperplanes in $\mathbb{R}^d$ and a subset $X \subseteq \mathbb{R}^{d-1}$, how does one compute efficiently the shortest vertical segment, with a vertical projection in X, that stabs (at least) $n-k$ of the hyperplanes of $\mathcal{H}$? In approximate form, one would like to find a vertical segment, with a vertical projection in X, that stabs at least $n-k$ of the hyperplanes of $\mathcal{H}$ and is of length at most a factor $(1+\varepsilon)$ of that of the optimal segment. (Note that linear programming with violations [22] can be applied directly when $X = \mathbb{R}^{d-1}$ or X is a convex polytope, as the problem can then be formulated as an LP-type problem. For our applications, however, X arises from our linearization technique and is not necessarily a convex set. In particular, the set X is a semi-algebraic set of constant complexity.) The direct approach for solving this problem involves computing the bottom/top k levels of $\mathcal{A}(\mathcal{H})$ and enumerating by brute force all possible such vertical segments (with vertical projection in X), where $\mathcal{A}(\mathcal{H})$ denotes the arrangement of hyperplanes of $\mathcal{H}$. The approach implemented in this fashion is doomed, as it leads to an inefficient algorithm with running time (roughly) of $O(n^d k^d)$. At the same time, since the k-level of $\mathcal{A}(\mathcal{H})$ is not necessarily convex for $k > 1$, we can no longer take advantage of known convex shape approximation techniques as was done in [3, 4, 19].

Instead, we study the question of approximating the first/last k levels directly. Note that the *combinatorial* approximation of k levels is relatively well understood (see [22, 2, 18]) and can be performed by random sampling or cuttings. (In the combinatorial settings, we are interested in finding a curve γ that lies close to the k-level under the crossing metric. Namely, any vertical segment connecting a point on γ to the k-level crosses at most εn lines.) However, our notion of approximation is stronger as it combines both the geometry and the combinatorial structure of the levels. In particular, we show that there is a coreset for this problem; namely, one can compute a

[1] In all our discussions, we consider the dimension to be a small constant, and as such the $O(\cdot)$ notation hides constants that depend solely on d.

subset of $O(k/\varepsilon^{d-1})$ hyperplanes such that the rth level of the coreset ε-approximates the r-level of the arrangement $\mathcal{A}(\mathcal{H})$ for $1 \le r \le k$. Here the approximation is the Euclidean distance between the level and the approximate level, compared with the length of the shortest vertical segment that connects the k-level with the $n-k$ level of the arrangement $\mathcal{A}(\mathcal{H})$. This can be done in $O(n + k/\varepsilon^{d-1})$ time. (All algorithms in this paper are randomized and their results are correct with high probability.)

We can apply the above results to a large number of shape fitting problems, including those considered in [3, 4, 19]. To name a few, we can approximate the following measures with k outliers: diameter, width, projection width, minimum enclosing ball, minimum-width annulus, minimum-volume spherical shell, minimum-width cylindrical shell, and also those measures for moving points. Note that most of those optimization problems fall outside the paradigm of linear programming and as such can not be solved using linear programming with violations. We can further extend our technique to handle insertions and deletions with polylogarithmic time per update.

In section 3, we present an algorithm to compute a small coreset for a set of hyperplanes in $\mathbb{R}^d$. We extend the results for a set of polynomials or their roots in section 4. In section 5, we present our results for various shape fitting problems with outliers. Finally, we conclude with section 6.

2. Preliminaries. Throughout the paper, we refer to the x_d-parallel direction in $\mathbb{R}^d$ as *vertical*. Given a point $x = (x_1, \ldots, x_{d-1})$ in $\mathbb{R}^{d-1}$, let (x, x_d) denote the point $(x_1, \ldots, x_{d-1}, x_d)$ in $\mathbb{R}^d$. Each point $x \in \mathbb{R}^d$ is also a vector in $\mathbb{R}^d$. Given a geometric object A, $A + x$ represents the object obtained by translating each point in A by x.

A *surface* is a subset of $\mathbb{R}^d$ that intersects any vertical line in a single point. Let A and B be either a point, a hyperplane, or a surface in $\mathbb{R}^d$. We say that A lies *above* (resp., *below*) B, denoted by $A \succeq B$ (resp., $A \preceq B$), if for any vertical line ℓ intersecting both A and B, we have that $x_d \ge y_d$ (resp., $x_d \le y_d$), where $(x_1, \ldots, x_{d-1}, x_d) = A \cap \ell$ and $(x_1, \ldots, x_{d-1}, y_d) = B \cap \ell$.

DEFINITION 2.1. *For a set of n hyperplanes $\mathcal{H}$ in $\mathbb{R}^d$, the* level *of a point $x \in \mathbb{R}^d$ in the arrangement $\mathcal{A}(\mathcal{H})$ is the number of hyperplanes of $\mathcal{H}$ lying vertically below x. For $k = 0, \ldots, n-1$, let $\mathbf{L}_{\mathcal{H},k}$ represent the surface which is the closure of all points on the hyperplanes of $\mathcal{H}$ whose level is k. We define the* top k-level *of $\mathcal{H}$ to be $\mathbf{U}_{\mathcal{H},k} = \mathbf{L}_{\mathcal{H},n-k-1}$ for $k = 0, \ldots, n-1$. Let $\mathbf{L}_{\mathcal{H},\le k}$ and $\mathbf{U}_{\mathcal{H},\le k}$ denote $\cup_{i=0}^{k} \mathbf{L}_{\mathcal{H},i}$ and $\cup_{i=0}^{k} \mathbf{U}_{\mathcal{H},i}$, respectively. Note that both $\mathbf{L}_{\mathcal{H},k}$ and $\mathbf{L}_{\mathcal{H},\le k}$ are subsets of the arrangement of $\mathcal{H}$. For $x \in \mathbb{R}^{d-1}$, we slightly abuse notation and define $\mathbf{L}_{\mathcal{H},k}(x)$ to be the value x_d such that $(x, x_d) \in \mathbf{L}_{\mathcal{H},k}$.*

DEFINITION 2.2. *The (k,r)-*extent *$\mathcal{H}|_r^k : \mathbb{R}^{d-1} \to \mathbb{R}$ is defined as the vertical distance between the r-level and the top k-level of $\mathcal{A}(\mathcal{H})$, i.e., for any $x \in \mathbb{R}^{d-1}$,*

$$\mathcal{H}|_r^k(x) = \mathbf{U}_{\mathcal{H},k}(x) - \mathbf{L}_{\mathcal{H},r}(x).$$

*The k-*extent *of $\mathcal{H}$ is the (k,k)-extent of $\mathcal{H}$ and is denoted by $\mathcal{E}_{\mathcal{H},k} = \mathcal{H}|_k^k$.*

DEFINITION 2.3. *Given a parameter $\varepsilon > 0$, a function $\mathcal{E}_\varepsilon : \mathbb{R}^{d-1} \to \mathbb{R}$ is an ε-*approximation *to $\mathcal{H}|_r^k(\cdot)$ if, for any $x \in \mathbb{R}^{d-1}$, we have $(1-\varepsilon)\mathcal{H}|_r^k(x) \le \mathcal{E}_\varepsilon(x) \le \mathcal{H}|_r^k(x)$.*

Our approximation relies on the idea that one can find a small subset of hyperplanes, which are "crucial" as far as the top/bottom k levels are concerned.

DEFINITION 2.4. *For a point $x \in \mathbb{R}^{d-1}$, a subset of hyperplanes $\mathcal{H}' \subseteq \mathcal{H}$ in $\mathbb{R}^d$ is*

a (k, ε, δ)-coreset at x *if the following holds: for any* $r \leq k$*, we have*

$$\mathbf{L}_{\mathcal{H},r}(x) \leq \mathbf{L}_{\mathcal{H}',r}(x) \leq \mathbf{L}_{\mathcal{H},r}(x) + \varepsilon \mathcal{E}_{\mathcal{H},k}(x) + \delta$$

and

$$\mathbf{U}_{\mathcal{H},r}(x) - \varepsilon \mathcal{E}_{\mathcal{H},k}(x) - \delta \leq \mathbf{U}_{\mathcal{H}',r}(x) \leq \mathbf{U}_{\mathcal{H},r}(x).$$

If $\mathcal{H}'$ *is a* (k, ε, δ)*-coreset at all points of* $\mathbb{R}^{d-1}$*, then it is* (k, ε, δ)-coreset *for* $\mathcal{H}$*.*

We omit δ from the above notation when $\delta = 0$. One of the main contributions of this paper is showing the existence of (k, ε)-coresets of size $O(k/\varepsilon^{d-1})$ for a set of hyperplanes in $\mathbb{R}^d$ for any given $k, \varepsilon > 0$. Note that if $\mathcal{H}'$ is a $(k, \varepsilon/2)$-coreset for $\mathcal{H}$, then the (r, t)-extent of $\mathcal{H}'$ ε-approximates that of $\mathcal{H}$ for any $r, t \leq k$. Therefore, in order to compute an ε-approximation of $\mathcal{E}_{\mathcal{H},k}$, one just needs to compute $\mathbf{L}_{\mathcal{H}',k}$ and $\mathbf{U}_{\mathcal{H}',k}$, and the resulting $\mathcal{E}_{\mathcal{H}',k}$ is the required approximation.

3. (k, ε)-coresets. In this section, given a set $\mathcal{H} = \{h_1, \ldots, h_n\}$ of hyperplanes in $\mathbb{R}^d$, and parameters $\varepsilon > 0$ and $1 \leq k \leq n$, we show how to compute a small (k, ε)-coreset for $\mathcal{H}$. Let $\Delta^{opt}(\mathcal{H}, k)$ denote the shortest vertical distance between the k-level and the top k-level of $\mathcal{A}(\mathcal{H})$, i.e., $\Delta^{opt}(\mathcal{H}, k) = \min_{x \in \mathbb{R}^{d-1}} \mathcal{E}_{\mathcal{H},k}(x)$. The algorithm first computes a set of $O(1)$ vertical segments, the length of each being at most $\Delta^{opt}(\mathcal{H}, k)$, such that, with high probability, all but $O(k)$ hyperplanes of $\mathcal{H}$ intersect at least one of those segments. Next, we subdivide $\mathcal{H}$ into $O(1/\varepsilon)$ disjoint subsets and show that the union of the $(k, \varepsilon, \varepsilon\Delta^{opt}(\mathcal{H}, k))$-coreset of each subset forms a (k, ε)-coreset for $\mathcal{H}$. Furthermore, we show that a coreset of size $O(k/\varepsilon^{d-2})$ can be computed efficiently for each subset, resulting in a coreset of size $O(k/\varepsilon^{d-1})$ for $\mathcal{H}$.

3.1. Short segments.

LEMMA 3.1. *Let* $\mathcal{J}$ *be a set of* n *hyperplanes in* $\mathbb{R}^d$*. In* $O(n)$ *time, one can compute a set* S *of* $O(1)$ *vertical segments such that with high probability* (i) *all the segments of* S *are no longer than* $\Delta^{opt}(\mathcal{J}, k)$*, and* (ii) $|\mathcal{J}_0| = O(k)$*, where* $\mathcal{J}_0$ *denotes the set of all hyperplanes of* $\mathcal{J}$ not *stabbed by the segments of* S*.*

Proof. The algorithm is iterative. Let $\mathcal{Q}_1 = \mathcal{J}$, $S_1 = \emptyset$. For $i > 1$, let S_i be the set of vertical segments computed in the beginning of the ith iteration. Let $\mathcal{Q}_i = \mathcal{J} \setminus S_i$ and $m_i = |\mathcal{Q}_i|$, where $\mathcal{J} \setminus S_i$ denotes the set of all hyperplanes of $\mathcal{J}$ not intersecting any segment of S_i. That is, $\mathcal{Q}_i$ is the set of hyperplanes not yet handled by the algorithm.

If $m_i = O(k)$, we are done, and set $\mathcal{J}_0$ to be $\mathcal{Q}_i$. Otherwise, if $m_i \leq n^{1/(3d)}$, we compute the k-level and the top k-level of the arrangement $\mathcal{A}(\mathcal{Q}_i)$ and the shortest vertical segment between those two levels. Let h_i^* denote this segment; then $|h_i^*| \leq \Delta^{opt}(\mathcal{J}, k)$. Set $S_{i+1} = S_i \cup \{h_i^*\}$ and observe that S_{i+1} is the required set, as $|\mathcal{J} \setminus S_{i+1}| = 2k$. Clearly, a naive implementation of the algorithm for this step would take $O(m_i^{2d+1}) = O(n)$ time.

Otherwise, if $m_i > n^{1/(3d)}$, then let $\mathcal{R}_i$ be a random sample from $\mathcal{Q}_i$ of size $O(n^{1/3d} \log n)$. Now consider the range space $\Sigma = (\mathcal{J}, \mathcal{X})$, where $\mathcal{X}$ consists of all subsets of $X \subseteq \mathcal{J}$ where there is a vertical segment s such that $X = s \cap \mathcal{J}$, where $s \cap \mathcal{J}$ denote the hyperplanes of $\mathcal{J}$ that intersect s. The size of $\mathcal{X}$ is bounded by $O(n^{d+1})$. Furthermore, the range space Σ has a bounded Vapnik–Chervonenkis dimension (VC-dimension). Therefore, by the ε-sample theorem [6], with high probability, $\mathcal{R}_i$ is an τ-sample of $\mathcal{Q}_i$, where $\tau = 1/n^{1/6d}$. That is, for any vertical segment s, with high probability, we have

$$\frac{|\mathcal{Q}_i \cap s|}{|\mathcal{Q}_i|} - \tau \leq \frac{|\mathcal{R}_i \cap s|}{|\mathcal{R}_i|} \leq \frac{|\mathcal{Q}_i \cap s|}{|\mathcal{Q}_i|} + \tau,$$

where $X \cap s$ denotes the set of hyperplanes of X that intersects s. Compute the arrangement $\mathcal{A}(\mathcal{R}_i)$, and compute the shortest vertical segment h_i^* between the K_i-level and the top K_i-level of $\mathcal{A}(\mathcal{R}_i)$, where

$$K_i = \left\lceil |\mathcal{R}_i| \left(\frac{k}{|\mathcal{Q}_i|} + \tau \right) \right\rceil.$$

Since $\mathcal{R}_i$ is a τ-sample of $\mathcal{Q}_i$, we have, with high probability, that

$$\mathbf{L}_{\mathcal{Q}_i,k} \preceq \mathbf{L}_{\mathcal{R}_i,K_i} \preceq \mathbf{U}_{\mathcal{R}_i,K_i} \preceq \mathbf{U}_{\mathcal{Q}_i,k}.$$

Thus, with high probability, the segment h_i^* is shorter than $\Delta^{opt}(\mathcal{J},k)$, and h_i^* intersects at least $m_i(1-4\tau)-2k$ hyperplanes of $\mathcal{Q}_i$. Let $S_{i+1} = S_i \cup \{h_i^*\}$, and let $\mathcal{Q}_{i+1} = \mathcal{Q}_i \setminus \{h_i^*\}$. This step takes $O(|\mathcal{R}_i|^{2d+1}) = O(n)$ time.

We are left with the task of bounding the number of iterations of the algorithm. Clearly, with high probability, $m_{i+1} \leq 4\tau m_i + 2k$. Therefore, for $m_i \geq 4k/\tau$, we have $m_{i+1} \leq 8\tau m_i \leq 8m_i/n^{1/6d}$. On the other hand, if $m_i \leq 4k/\tau$, then $m_{i+1} \leq 6k$ and the algorithm stops in the next iteration. We conclude that the number of iterations performed by the algorithm is $O(\log_{1/\tau} n) = O(6d) = O(1)$ with high probability, which implies the lemma. □

The algorithm of Lemma 3.1 can be derandomized using deterministic construction of ε-samples [9]. This results in a somewhat slower algorithm. Note that the algorithm of Lemma 3.1 can be modified to output, together with each vertical segment of S, the corresponding set of hyperplanes of $\mathcal{J}$ that intersect it within the same time bound.

3.2. Disjoint subsets. Apply Lemma 3.1 to $\mathcal{H}$. Let S denote the set of segments generated, and let $\mathcal{H}_0$ be the set of hyperplanes of $\mathcal{H}$ not stabbed by any segment of S. Break every segment of S into $\lceil 4/\varepsilon \rceil$ equal-length subsegments. For any such subsegment s, we have $|s| \leq \varepsilon \Delta^{opt}(\mathcal{H},k)/4$. Let $S' = \{s_1, \ldots, s_m\}$ denote the resulting set of segments, where $m = O(1/\varepsilon)$ with high probability. We distribute the hyperplanes of $\mathcal{H} \setminus \mathcal{H}_0$ into the sets associated with the subsegment of S' that they intersect. If a hyperplane is stabbed by more than one segment from S', it is only distributed to (an arbitrary) one of them. Let $\mathcal{H}_i$ be the resulting set of hyperplanes associated with segment s_i for $i = 1, \ldots, m$. By construction $|\mathcal{H}| = \sum_{i=0}^{m} |\mathcal{H}_i|$, and $|\mathcal{H}_0| = O(k)$. This redistribution process can be easily performed in linear time by using the floor function and by observing that each of the hyperplanes of $\mathcal{H} \setminus \mathcal{H}_0$ is already associated with one of the segments of S.

DEFINITION 3.2. *A set of hyperplanes $\mathcal{J}$ is a* δ-sheaf *if there exists a vertical segment s of length δ such that all the hyperplanes of $\mathcal{J}$ stab s. The vertical segment s is the* axis *of $\mathcal{J}$.*[2] *A 0-sheaf is referred to as a* sheaf*; in this case all hyperplanes in $\mathcal{J}$ pass through a common point, which is referred to as the* focal point *of $\mathcal{J}$. See* Figure 3.1.

Using the above splitting approach, together with Lemma 3.1, implies the following theorem.

THEOREM 3.3. *Given a set $\mathcal{J}$ of n hyperplanes in $\mathbb{R}^d$ and parameters $k, \varepsilon > 0$, one can compute, in $O(n + 1/\varepsilon)$ time, a partition of $\mathcal{J}$ into $m = O(1/\varepsilon)$ sets, $\mathcal{J}_0, \ldots, \mathcal{J}_m$, such that, with high probability, $|\mathcal{J}_0| = O(k)$, and $\mathcal{J}_i$ is an $(\varepsilon \Delta^{opt}(\mathcal{J},k))$-sheaf for $i = 1, \ldots, m$.*

[2] We will refer to it as the *axis of evil* when appropriate.

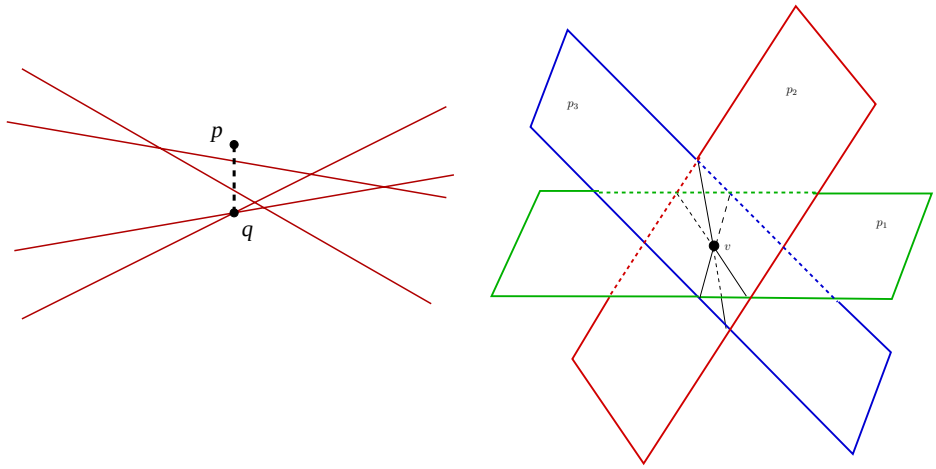


FIG. 3.1. (a) *A δ-sheaf in $\mathbb{R}^2$ with axis pq.* (b) *A sheaf in $\mathbb{R}^3$ with focal point v.*

LEMMA 3.4. *Let $\mathcal{J} = \cup_{i=0}^{m} \mathcal{J}_i$, and let $\mathcal{J}' = \cup_{i=0}^{m} \mathcal{J}'_i$, where $\mathcal{J}'_i \subseteq \mathcal{J}_i$ is a (k, ε, δ)-coreset for $\mathcal{J}_i$ for $i = 0, \ldots, m$. Then $\mathcal{J}'$ is a (k, ε, δ)-coreset for $\mathcal{J}$.*

Proof. Fix any $r \leq k$ and consider any point $x \in \mathbb{R}^{d-1}$. Let $p = (x, \mathbf{L}_{\mathcal{J},r}(x))$, and let $i_0, \ldots, i_m$ be the number of hyperplanes lying below p in $\mathcal{J}_0, \ldots, \mathcal{J}_m$, respectively, such that $i_0 + \cdots + i_m = r$. Observe that $\mathbf{L}_{\mathcal{J},r}(x) = \max_j \mathbf{L}_{\mathcal{J}_j,i_j}(x)$ and that for any subset $\mathcal{Q} \subseteq \mathcal{J}$, we have $\mathbf{L}_{\mathcal{J},r}(x) \leq \mathbf{L}_{\mathcal{Q},r}(x)$ and $\mathbf{U}_{\mathcal{J},r}(x) \geq \mathbf{U}_{\mathcal{Q},r}(x)$, implying that $\mathcal{E}_{\mathcal{Q},k}(x) \leq \mathcal{E}_{\mathcal{J},k}(x)$. Therefore,

$$\begin{aligned}
\mathbf{L}_{\mathcal{J},r}(x) &\leq \mathbf{L}_{\mathcal{J}',r}(x) \leq \max_{j=1}^{m} \mathbf{L}_{\mathcal{J}'_j,i_j}(x) \\
&\leq \max_{j=1}^{m} \big(\mathbf{L}_{\mathcal{J}_j,i_j}(x) + \varepsilon \mathcal{E}_{\mathcal{J}_j,k}(x) + \delta \big) \\
&\leq \left(\max_{j=1}^{m} \mathbf{L}_{\mathcal{J}_j,i_j}(x) \right) + \varepsilon \mathcal{E}_{\mathcal{J},k}(x) + \delta \\
&= \mathbf{L}_{\mathcal{J},r}(x) + \varepsilon \mathcal{E}_{\mathcal{J},k}(x) + \delta.
\end{aligned}$$

A symmetric argument proves the claim for $\mathbf{U}_{\mathcal{J},r}(x)$. ∎

The above lemma implies that if we can compute a (k, ε, δ)-coreset of a small size for each $\mathcal{H}_i$, then we can easily compute such a coreset for $\mathcal{H}$.

LEMMA 3.5. *For a set $\mathcal{J}$ of n hyperplanes in $\mathbb{R}^d$ and a parameter $\delta > 0$, let $\underline{\mathcal{J}}$ be the set of hyperplanes resulting by translating each hyperplane h of $\mathcal{J}$ downward by a vertical distance δ_h, where $0 \leq \delta_h \leq \delta$. Then for any $x \in \mathbb{R}^{d-1}$ and $1 \leq r, k \leq n$, we have*

(i) $\mathbf{L}_{\mathcal{J},r}(x) - \delta \leq \mathbf{L}_{\underline{\mathcal{J}},r}(x) \leq \mathbf{L}_{\mathcal{J},r}(x)$ *and* $\mathbf{U}_{\mathcal{J},r}(x) - \delta \leq \mathbf{U}_{\underline{\mathcal{J}},r}(x) \leq \mathbf{U}_{\mathcal{J},r}(x)$.
(ii) $\mathcal{E}_{\mathcal{J},k}(x) - \delta \leq \mathcal{E}_{\underline{\mathcal{J}},k}(x) \leq \mathcal{E}_{\mathcal{J},k}(x) + \delta$.

Proof. The second part follows immediately from the first one. As for the first claim, observe that $\mathbf{L}_{\underline{\mathcal{J}},r}(x) \leq \mathbf{L}_{\mathcal{J},r}(x)$ holds since all the planes of $\underline{\mathcal{J}}$ are copies of hyperplanes of $\mathcal{J}$ which were translated downward.

Next, fix any $x \in \mathbb{R}^{d-1}$, and let $h_1, \ldots, h_{n-r}$ be the hyperplanes of $\mathcal{J}$ lying above $\mathbf{L}_{\mathcal{J},r}(x)$, and let $\underline{h}_0, \ldots, \underline{h}_{n-r}$ be the corresponding hyperplanes in $\underline{\mathcal{J}}$. Clearly, $\underline{h}_0, \ldots, \underline{h}_{n-r}$ lie above point $(x, \mathbf{L}_{\mathcal{J}}(x) - \delta)$, as each hyperplane was translated down

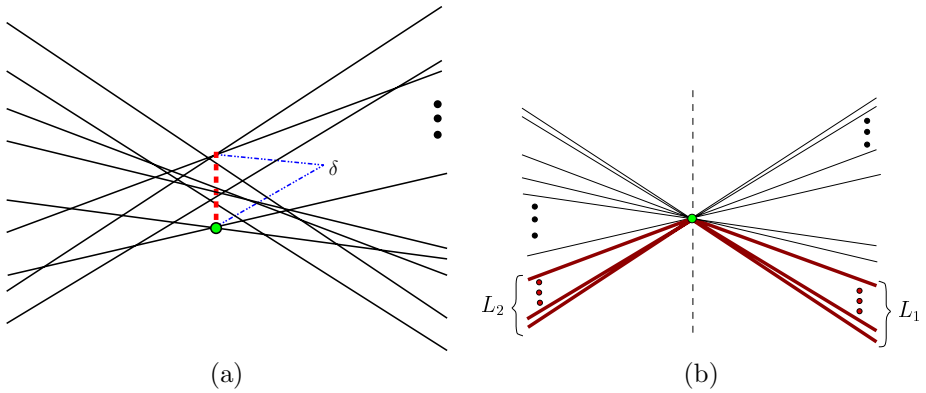


FIG. 3.2. (a) *L is a δ-sheaf. By shifting all lines downward we have $\underline{L}$ in* (b). *$\underline{L}'$ is composed of L_1, the set of k lines with smallest slope; and L_2, the set of k lines with largest slope. Thick segments form the $\leq k$-levels of L.*

by a distance at most δ. Thus $\mathbf{L}_{\mathcal{J},r}(x) - \delta \leq \mathbf{L}_{\underline{\mathcal{J}},r}(x)$. ∎

Observation 3.6. Given a set $\mathcal{H}$ of hyperplanes in $\mathbb{R}^d$, and a (k,ε,δ)-coreset $\mathcal{H}'$ for $\mathcal{H}$, the set $\mathcal{H}'$ is a (k,μ)-coreset for $\mathcal{H}$, where $\mu = \varepsilon + \delta/\Delta^{opt}(\mathcal{H},k)$.

3.3. The two-dimensional case.

LEMMA 3.7. *Let L be a set of n lines in the plane which is a δ-sheaf. Given $k > 0$, one can compute, in $O(n+k)$ time, a subset $L' \subseteq L$ such that* (i) $|L'| = 2k$, *and* (ii) *L' is a $(k,0,\delta)$-coreset for L.*

Proof. Translate each line in L downward so that it passes through the point q, where q is the bottom endpoint of the axis of the δ-sheaf. Call the resulting set $\underline{L}$; $\underline{L}$ is a sheaf in the plane with focal point q. Let $\underline{L}'$ be the union of the set of k lines of $\underline{L}$ with the largest slope and the set of k lines of $\underline{L}$ with the smallest slope. It is easy to verify that $\underline{L}'$ is a $(k,0)$-coreset for $\underline{L}$ (i.e., $\mathcal{A}(\underline{L})$ and $\mathcal{A}(\underline{L}')$ have the same top and bottom k levels), and $\left|\underline{L}'\right| = 2k$. See Figure 3.2.

Let $L' \subseteq L$ be the set of original lines that corresponds to the lines of $\underline{L}'$. By Lemma 3.5, for any $x \in \mathbb{R}$ and $0 \leq r \leq k$, we have

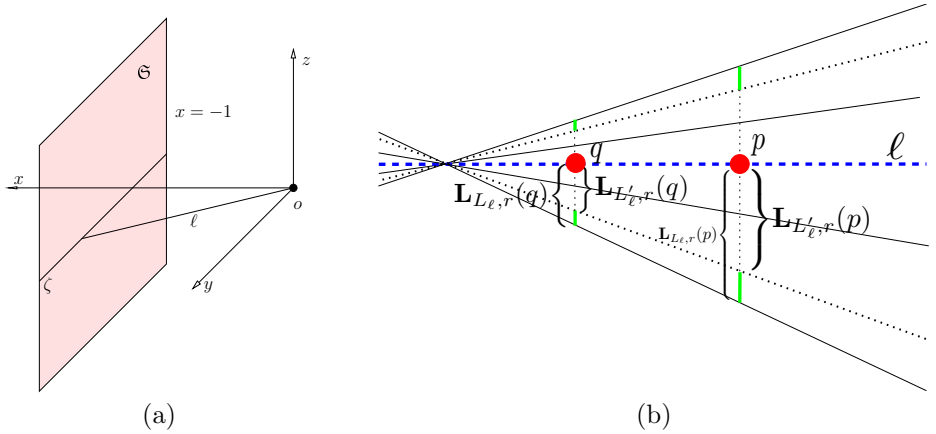
$$\mathbf{L}_{L,r}(x) \leq \mathbf{L}_{L',r}(x) \leq \mathbf{L}_{\underline{L}',r}(x) + \delta = \mathbf{L}_{\underline{L},r}(x) + \delta \leq \mathbf{L}_{L,r}(x) + \delta$$

and

$$\mathbf{U}_{L,r}(x) \geq \mathbf{U}_{L',r}(x) \geq \mathbf{U}_{\underline{L}',r}(x) = \mathbf{U}_{\underline{L},r}(x) \geq \mathbf{U}_{L,r}(x) - \delta. \quad ∎$$

THEOREM 3.8. *Given a set L of n lines in the plane and parameters $k, \varepsilon > 0$, one can compute, in $O(n + k/\varepsilon)$ time, with high probability, a set $L' \subseteq L$ such that* (i) *L' is a (k,ε)-coreset for L, and* (ii) $|L'| = O(k/\varepsilon)$.

Proof. Apply Theorem 3.3 to L, and let $L = \cup_{i=0}^{m} L_i$ be the resulting set, where $m = O(1/\varepsilon)$. With high probability, the sets $L_1, \ldots, L_m$ are δ-sheaves, and L_0 has size $O(k)$, where $\delta = \varepsilon\Delta^{opt}(L,k)$. Apply Lemma 3.7 to each L_i for $i = 1, \ldots, m$, and let $L_i' \subseteq L_i \subseteq L$ be the resulting $(k,0,\delta)$-coreset. It then follows from Lemma 3.4 that $L' = L_0 \cup \bigcup_{i=1}^{m} L_i'$ is a $(k,0,\delta)$-coreset for L. By Observation 3.6 the set L' is a (k,ε)-coreset for L. ∎



(a) (b)

FIG. 3.3. (a) *Illustration of the plane* $\mathfrak{S}$ *and line* ζ*;* (b) *The set of lines* L'_ℓ *(dashed lines) is a* (k,ε)*-coreset for the set of lines* L_ℓ *if and only if for an arbitrary point* p *on* ℓ*, which is different from the origin,* L'_ℓ *is a* (k,ε)*-coreset at* p*.*

3.4. The three-dimensional case.

LEMMA 3.9. *Given a sheaf* $\mathcal{J}$ *of* n *planes in three dimensions with its focal point at the origin* o *and parameters* k*,* $0<\varepsilon<1$*, one can compute, in* $O(n+k/\varepsilon)$ *time, a subset* $\mathcal{J}'$ *such that, with high probability,* $\mathcal{J}'$ *is a* (k,ε)*-coreset for* $\mathcal{J}$ *and* $|\mathcal{J}'|=O(k/\varepsilon)$*.*

Proof. Consider the plane $\mathfrak{S}\equiv(x=-1)\equiv\{(-1,y,z)\;\big|\;y,z\in\mathbb{R}\}$. Let $\zeta\subseteq\mathfrak{S}$ be the line

$$(x=-1)\cap(z=0)\equiv\left\{(-1,y,0)\;\middle|\;y\in\mathbb{R}\right\},$$

that is, the intersection between plane $\mathfrak{S}$ and the xy-plane (see Figure 3.3(a)). Consider the set L_ζ of lines obtained by intersecting $\mathfrak{S}$ with the planes of $\mathcal{J}$. By Theorem 3.8, one can compute a subset $L'_\zeta\subseteq L_\zeta$ of size $O(k/\varepsilon)$ which is a (k,ε)-coreset for L_ζ. Let $\mathcal{J}'$ be set of planes of $\mathcal{J}$ corresponding to L'_ζ. We claim that $\mathcal{J}'$ is a (k,ε)-coreset for $\mathcal{J}$ at any point in the xy-plane, which implies that it is a (k,ε)-coreset for $\mathcal{J}$.

Indeed, let ℓ be any line on the xy-plane that passes through the origin, and let h_ℓ be the vertical plane perpendicular to the xy-plane, with ℓ being its intersection with the xy-plane. Let L_ℓ be the set of lines formed by the intersection of h_ℓ and planes of $\mathcal{J}$. Clearly, L_ℓ form a sheaf structure on h_ℓ, as illustrated in Figure 3.3(b).

We claim that if $\mathcal{J}'\subset\mathcal{J}$ is a (k,ε)-coreset at p for any $p\in\ell$ (different from the origin), then it is (k,ε)-coreset at all points in ℓ. Indeed, first observe that $\mathbf{L}_{L_\ell,r}(p)=\mathbf{L}_{\mathcal{J},r}(p)$ and $\mathbf{L}_{L'_\ell,r}(p)=\mathbf{L}_{\mathcal{J}',r}(p)$, where L'_ℓ is the set of lines induced by the intersection of h_ℓ with planes of $\mathcal{J}'$. Now since $\mathcal{J}'$ is a (k,ε)-coreset at p, we have

$$\mathbf{L}_{L_\ell,r}(p)\le\mathbf{L}_{L'_\ell,r}(p)\le\mathbf{L}_{L_\ell,r}(p)+\varepsilon\mathcal{E}_{L_\ell,k}(p)$$

for any $1\le r\le k$, which implies that, for any $q\in\ell$ (without loss of generality, we assume that q and p are on the same side of the origin on ℓ), we have $\mathbf{L}_{L_\ell,r}(q)\le\mathbf{L}_{L'_\ell,r}(q)\le\mathbf{L}_{L_\ell,r}(q)+\varepsilon\mathcal{E}_{L_\ell,k}(q)$. The last step follows from similarity of triangles since

$$\frac{\mathbf{L}_{L_\ell,r}(p)}{\mathbf{L}_{L_\ell,r}(q)}=\frac{\mathbf{L}_{L'_\ell,r}(p)}{\mathbf{L}_{L'_\ell,r}(q)}=\frac{\mathcal{E}_{L_\ell,k}(p)}{\mathcal{E}_{L_\ell,k}(q)},$$

as illustrated in Figure 3.3(b).

Now consider any line ℓ that passes through the origin and assume that it intersects ζ at point u. Recall that L'_ζ is a (k,ε)-coreset for L_ζ, implying that $\mathcal{J}'$ is a (k,ε)-coreset at point u. Hence, by the above discussion, $\mathcal{J}'$ is a (k,ε)-coreset for all points on ℓ. It then follows that $\mathcal{J}'$ is a (k,ε)-coreset for all points on the xy-plane, except the points lying on the y-axis. It is now straightforward to use a limit argument to show that $\mathcal{J}'$ is a (k,ε)-coreset for all the points in the xy-plane, as required.

Note that for the sake of simplicity, we argued only about $\mathbf{L}_{L_\ell,r}$ in the preceding discussion, since the same analysis holds for $\mathbf{U}_{L_\ell,r}$ by symmetry. □

LEMMA 3.10. *Given a δ-sheaf $\mathcal{J}$ of n planes in three dimensions and parameters k, $0<\varepsilon<1$, one can compute, in $O(n+k/\varepsilon)$ time, a subset $\mathcal{J}'$ such that, with high probability, $\mathcal{J}'$ is a $(k,\varepsilon,2\delta)$-coreset for $\mathcal{J}$, and $|\mathcal{J}'|=O(k/\varepsilon)$.*

Proof. Translate the planes of $\mathcal{J}$ downward by a distance at most δ such that they all pass through o, which is the bottom vertex of the axis of $\mathcal{J}$. Let $\underline{\mathcal{J}}$ denote the resulting sheaf, and assume that the focal point o of $\underline{\mathcal{J}}$ is in the origin. By Lemma 3.9, one can compute a subset $\underline{\mathcal{J}'}\subseteq\underline{\mathcal{J}}$, which is a (k,ε)-coreset for $\underline{\mathcal{J}}$, $\left|\underline{\mathcal{J}'}\right|=O(k/\varepsilon)$, and this set can be computed in $O(n+k/\varepsilon)$ time.

Let $\mathcal{J}'$ be the set of original planes that corresponds to the planes of $\underline{\mathcal{J}'}$. Then $\mathcal{J}'$ is a $(k,\varepsilon,2\delta)$-coreset for $\mathcal{J}$. Indeed, by Lemma 3.5, for any $p\in\mathbb{R}^2$,

$$\begin{aligned}\mathbf{L}_{\mathcal{J},r}(p)&\le\mathbf{L}_{\mathcal{J}',r}(p)\le\mathbf{L}_{\underline{\mathcal{J}'},r}(p)+\delta\le\mathbf{L}_{\underline{\mathcal{J}},r}(p)+\varepsilon\mathcal{E}_{\underline{\mathcal{J}},k}(p)+\delta\\&\le\mathbf{L}_{\mathcal{J},r}(p)+\varepsilon\mathcal{E}_{\underline{\mathcal{J}},k}(p)+\delta\le\mathbf{L}_{\mathcal{J},r}(p)+\varepsilon\mathcal{E}_{\mathcal{J},k}(p)+2\delta,\end{aligned}$$

since $\mathcal{E}_{\underline{\mathcal{J}},k}(p)\le\mathcal{E}_{\mathcal{J},k}(p)+\delta$ and $\varepsilon<1$. □

THEOREM 3.11. *Given a set $\mathcal{H}$ of n planes in three dimensions and parameters k,ε, one can compute, in $O(n+k/\varepsilon^2)$ time, a subset $\mathcal{H}'\subseteq\mathcal{H}$ such that, with high probability, it holds that* (i) *$\mathcal{H}'$ is a (k,ε)-coreset for $\mathcal{H}$, and* (ii) *$|\mathcal{H}'|=O(k/\varepsilon^2)$.*

Proof. Apply Theorem 3.3 to $\mathcal{H}$ with $\varepsilon/4$ and k. This results in a partition of $\mathcal{H}$ into sets $\mathcal{H}_0,\mathcal{H}_1,\ldots,\mathcal{H}_m\subseteq\mathcal{H}$, where $\mathcal{H}_1,\ldots,\mathcal{H}_m$ are δ-sheafs, $\delta=(\varepsilon/4)\Delta^{opt}(\mathcal{H},k)$, $m=O(1/\varepsilon)$, and $|\mathcal{H}_0|=O(k)$ with high probability.

Applying Lemma 3.10 to $\mathcal{H}_i$ for $i=1,\ldots,m$ results in a $(k,\varepsilon/2,2\delta)$-coreset $\mathcal{H}'_i$ for $\mathcal{H}_i$, where $|\mathcal{H}'_i|=O(k/\varepsilon)$.

Let $\mathcal{H}'=\mathcal{H}_0\cup\bigcup_{i=1}^m\mathcal{H}'_i$. By Lemma 3.4, $\mathcal{H}'$ is a $(k,\varepsilon/2,2\delta)$-coreset for $\mathcal{H}$, which in turn implies that $\mathcal{H}'$ is a (k,ε)-coreset for $\mathcal{H}$, by Observation 3.6.

Finally, $|\mathcal{H}'|=|\mathcal{H}_0|+\sum_{i=1}^m|\mathcal{H}'_i|=O(k+(1/\varepsilon)(k/\varepsilon))=O(k/\varepsilon^2)$. The overall running time is $O(n+1/\varepsilon+n+k/\varepsilon^2)=O(n+k/\varepsilon^2)$. □

3.5. The higher-dimensional case. Let $f(n,k,\varepsilon,d)$ denote a bound on the size of (k,ε)-coreset of n hyperplanes in $\mathbb{R}^d$. We know that $f(n,k,\varepsilon,2)=O(k/\varepsilon)$ and $f(n,k,\varepsilon,3)=O(k/\varepsilon^2)$ by Theorems 3.8 and 3.11, respectively. Let $T(n,k,\varepsilon,d)$ denote the time needed to compute such a set.

The proof of the following lemma is a direct extension of the proof of Lemma 3.10.

LEMMA 3.12. *Given a δ-sheaf $\mathcal{J}$ of n hyperplanes in $\mathbb{R}^d$ and parameters $k,\varepsilon>0$, then one can compute, in $O(T(n,k,\varepsilon,d-1))$ time, a subset $\mathcal{J}'$ such that $\mathcal{J}'$ is a $(k,\varepsilon,2\delta)$-coreset for $\mathcal{J}$, and $|\mathcal{J}'|=f(n,k,\varepsilon,d-1)$.*

THEOREM 3.13. *Given a set $\mathcal{H}$ of n hyperplanes in $\mathbb{R}^d$ and parameters $k,\varepsilon>0$, with high probability, one can compute, in $T(n,k,\varepsilon,d)=O(n+k/\varepsilon^{d-1})$ time, a subset $\mathcal{H}'$ such that* (i) *$\mathcal{H}'$ is a (k,ε)-coreset for $\mathcal{H}$, and* (ii) *$f(n,k,\varepsilon,d)=|\mathcal{H}'|=O(k/\varepsilon^{d-1})$.*

Proof. The proof of the theorem is a straightforward extension of the proof of Theorem 3.11, and as such we verify only the bounds on the coreset size and running

time. We have

$$f(n,k,\varepsilon,d) = O(k) + O\left(\frac{1}{\varepsilon} f(n,k,\varepsilon,d-1)\right) = O\left(\frac{k}{\varepsilon^{d-1}}\right).$$

As for the running time, we have

$$T(n,k,\varepsilon,d) = O\left(n + \frac{1}{\varepsilon}\right) + \sum_{i=1}^{O(1/\varepsilon)} T(n_i,k,\varepsilon,d-1),$$

where $\sum_i n_i \leq n$ and $T(n,k,\varepsilon,3) = O(n+k/\varepsilon^2)$, by Theorem 3.11. Thus, $T(n,k,\varepsilon,d) = O(n+k/\varepsilon^{d-1})$, as claimed. ∎

4. Polynomials and their roots. In this section, we extend previous results to the extent of a family of polynomials and their roots.

4.1. Polynomials. Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ be a family of d-variate polynomials and let $u_1, \ldots, u_d$ be the variables over which the functions of $\mathcal{F}$ are defined. Each f_i corresponds to a surface in $\mathbb{R}^{d+1}$. For example, any d-variate linear function can be considered as a hyperplane in $\mathbb{R}^{d+1}$ (and vice versa). We extend, in the natural way, the definitions of k-level, (k,r)-extent, ε-approximation, and (k,ε,δ)-coreset for the arrangement $\mathcal{A}(\mathcal{F})$.

Each monomial over $u_1, \ldots, u_d$ appearing in $\mathcal{F}$ can be mapped to a distinct variable x_i. Let $x_1, \ldots, x_s$ be the resulting variables. As such $\mathcal{F}$ can be linearized into a set $\mathcal{H} = \{h_1, \ldots, h_n\}$ of linear functions over $\mathbb{R}^s$. In particular, $\mathcal{H}$ is a set of n hyperplanes in $\mathbb{R}^{s+1}$. Note that the surface induced by f_i in $\mathbb{R}^{d+1}$ corresponds only to a subset of the surface of h_i in $\mathbb{R}^{s+1}$. This technique is called *linearization* and has been widely used in fields such as machine learning [11] and computational geometry [5].

For example, consider a family of polynomials $\mathcal{F} = \{f_1, \ldots, f_n\}$, where $f_i(x,y) = a_i(x^2+y^2) + b_i x + c_i y + d_i$, and $a_i, b_i, c_i, d_i \in \mathbb{R}$ for $i = 1, \ldots, n$. This family of polynomials defined over $\mathbb{R}^2$ can be linearized to a family of linear functions defined over $\mathbb{R}^3$, by $h_i(x,y,z) = a_i z + b_i x + c_i y + d_i$, and setting $\mathcal{H} = \{h_1, \ldots, h_n\}$. Clearly, $\mathcal{H}$ is a set of hyperplanes in $\mathbb{R}^4$, and $f_i(x,y) = h_i(x,y,x^2+y^2)$. Thus, for any point $(x,y) \in \mathbb{R}^2$, instead of evaluating $\mathcal{F}$ on (x,y), we can evaluate $\mathcal{H}$ on $\eta(x,y) = (x,y,x^2+y^2)$, where $\eta(x,y)$ is the *linearization image* of (x,y). The advantage of this linearization is that $\mathcal{H}$, being a family of linear functions, is now easier to handle than $\mathcal{F}$. There is a general technique for finding the best possible linearization (i.e., a mapping η with the target dimension as small as possible); see [5] for details. Observe that $X = \eta(\mathbb{R}^2)$ is a *subset* of $\mathbb{R}^3$ (this is the "standard" paraboloid), and we are interested in the value of $\mathcal{H}$ only on points belonging to X. In particular, the set X is not necessarily convex. The set X resulting from the linearization is a semialgebraic set of constant complexity, and as such basic manipulation operations of X can be performed in constant time.

Note that for each $1 \leq i \leq n$, $f_i(p) = h_i(\eta(p))$ for $p \in \mathbb{R}^d$. As such, if $\mathcal{H}' \subseteq \mathcal{H}$ is a (k,ε)-coreset for $\mathcal{H}$, then clearly the corresponding subset in $\mathcal{F}$ is a (k,ε)-coreset for $\mathcal{F}$. The following theorem is a restatement of Theorem 3.13 in this settings.

THEOREM 4.1. *Given a family of d-variate polynomials $\mathcal{F} = \{f_1, \ldots, f_n\}$ and parameters k and ε, one can compute, in $O(n + k/\varepsilon^s)$ time, a subset $\mathcal{F}' \subseteq \mathcal{F}$ of $O(k/\varepsilon^s)$ polynomials such that $\mathcal{F}'$ is a (k,ε)-coreset for $\mathcal{F}$, with high probability. Here s is the number of different monomials present in the polynomials of $\mathcal{F}$.*

4.2. Roots of polynomials. It turns out that for roots of polynomials the definition of coreset (Definition 2.4) is somewhat too restrictive. However, we can get a similar notion of a coreset by slightly relaxing the requirements.

DEFINITION 4.2. *Let $\mathcal{F}$ be a set of nonnegative functions defined over $\mathbb{R}^d$. A subset $\mathcal{F}' \subseteq \mathcal{F}$ is* (k,ε)-sensitive *if, for any $r \leq k$ and $x \in \mathbb{R}^d$, we have*

$$\mathbf{L}_{\mathcal{F},r}(x) \leq \mathbf{L}_{\mathcal{F}',r}(x) \leq \mathbf{L}_{\mathcal{F},r}(x) + \frac{\varepsilon}{2}\mathcal{F}|_r^k(x),$$

$$\mathbf{U}_{\mathcal{F},r}(x) - \frac{\varepsilon}{2}\mathcal{F}|_k^r(x) \leq \mathbf{U}_{\mathcal{F}',r}(x) \leq \mathbf{U}_{\mathcal{F},r}(x).$$

Note that any (k,ε)-coreset is $(k,2\varepsilon)$-sensitive (while the inverse is not necessarily true).

LEMMA 4.3. *If $\mathcal{F}'$ is (k,ε)-sensitive for $\mathcal{F}$, then*

$$(1-\varepsilon)\,\mathcal{F}|_r^t(x) \leq \mathcal{F}'|_r^t(x) \leq \mathcal{F}|_r^t(x)$$

for any $t, r \leq k$ and $x \in \mathbb{R}^d$.

Proof.

$$\begin{aligned}
\mathcal{F}|_r^t(x) &\geq \mathcal{F}'|_r^t(x) = \mathbf{U}_{\mathcal{F}',t}(x) - \mathbf{L}_{\mathcal{F}',r}(x) \\
&\geq \Big(\mathbf{U}_{\mathcal{F},t}(x) - \frac{\varepsilon}{2}\mathcal{F}|_k^t(x)\Big) - \Big(\mathbf{L}_{\mathcal{F},r}(x) + \frac{\varepsilon}{2}\mathcal{F}|_r^k(x)\Big) \\
&= (\mathbf{U}_{\mathcal{F},t}(x) - \mathbf{L}_{\mathcal{F},r}(x)) - \frac{\varepsilon}{2}\Big(\mathcal{F}|_k^t(x) + \mathcal{F}|_r^k(x)\Big) \\
&\geq \mathcal{F}|_r^t(x) - \varepsilon\mathcal{F}|_r^t(x) = (1-\varepsilon)\,\mathcal{F}|_r^t(x). \qquad \square
\end{aligned}$$

THEOREM 4.4. *Let $\mathcal{F} = \{f_1^{1/2}, \ldots, f_n^{1/2}\}$ be a family of d-variate functions defined over $\mathbb{R}^d$, where f_i is a d-variate polynomial for $i = 1, \ldots, n$. Given k and $0 < \varepsilon < 1$, one can compute, in $O(n + k/\varepsilon^{2s})$ time, a subset $\mathcal{F}' \subseteq \mathcal{F}$ such that, with high probability, $\mathcal{F}'$ is (k,ε)-sensitive for $\mathcal{F}$ and $|\mathcal{F}'| = O(k/\varepsilon^{2s})$, where s is the number of distinct monomials present in the polynomials of $\mathcal{F}$.*

Proof. Let $\mathcal{G} = \{f_i \mid f_i^{1/2} \in \mathcal{F}\}$ denote the set of d-variate polynomials which are the square of the functions in $\mathcal{F}$. Let $\mathcal{H}$ be the set of hyperplanes in $\mathbb{R}^{s+1}$ obtained by linearization of f_i's, and let $\eta : \mathbb{R}^d \to \mathbb{R}^s$ be the linearization used. Given any $k, \varepsilon > 0$, one can compute in $O(n + k/\delta^s)$ time, a (k,δ)-coreset $\mathcal{H}' \subseteq \mathcal{H}$ of size $O(k/\delta^s)$ for $\mathcal{H}$, where $\delta = \varepsilon^2/32$. Clearly, $\mathcal{H}'$ is $(k,2\delta)$-sensitive for $\mathcal{H}$.

Let $\mathcal{G}' \subseteq \mathcal{G}$ be the set of functions of $\mathcal{G}$ that corresponds to $\mathcal{H}'$. Let α be any point in $\mathbb{R}^d$, and let $x = \eta(\alpha) \in \mathbb{R}^s$ be the linearized image of α. Let $a = \mathbf{L}_{\mathcal{G},r}(\alpha) = \mathbf{L}_{\mathcal{H},r}(x)$, $A = \mathbf{L}_{\mathcal{G}',r}(x) = \mathbf{L}_{\mathcal{H}',r}(x)$, $b = \mathbf{U}_{\mathcal{G},k}(\alpha) = \mathbf{U}_{\mathcal{H},k}(x)$. By the definition of $(k,2\delta)$-sensitivity, we have that $A - a \leq \delta\, \mathcal{G}|_r^k(x) = \delta(b-a)$.

Now, if $\sqrt{A} + \sqrt{a} \leq 2(\delta/\varepsilon)(\sqrt{b} + \sqrt{a})$, then $\sqrt{A} + \sqrt{a} \leq 4\sqrt{b}\delta/\varepsilon \leq \sqrt{b}\varepsilon/8$. Thus,

$$\begin{aligned}
\sqrt{A} - \sqrt{a} \leq \sqrt{A} + \sqrt{a} &\leq \frac{\sqrt{b}\varepsilon}{8} \leq \Big(\frac{\varepsilon}{2} - \frac{\varepsilon}{8}\Big)\sqrt{b} = \frac{\varepsilon}{2}\sqrt{b} - \frac{\varepsilon}{8}\sqrt{b} \leq \frac{\varepsilon}{2}\sqrt{b} - \sqrt{a} \\
&\leq \frac{\varepsilon}{2}(\sqrt{b} - \sqrt{a}),
\end{aligned}$$

since $\sqrt{a} \leq \sqrt{A} + \sqrt{a} \leq \sqrt{b}\varepsilon/8$.

Otherwise, $\sqrt{A}+\sqrt{a} \geq 2(\delta/\varepsilon)(\sqrt{b}+\sqrt{a})$ and

$$\begin{aligned}\mathbf{L}_{\mathcal{F}',r}(\alpha) - \mathbf{L}_{\mathcal{F},r}(\alpha) &= \sqrt{A}-\sqrt{a} = \frac{A-a}{\sqrt{A}+\sqrt{a}} \leq \frac{\delta(b-a)}{\sqrt{A}+\sqrt{a}} \leq \frac{\delta(b-a)}{2(\delta/\varepsilon)(\sqrt{b}+\sqrt{a})} \\ &= (\varepsilon/2)(\sqrt{b}-\sqrt{a}) \\ &\leq \frac{\varepsilon}{2}\mathcal{F}|_r^k(\alpha). \qquad \square\end{aligned}$$

5. Applications. In this section, we briefly present some of the results that follow from our technique for various shape fitting problems with outliers. Some of the other results mentioned in the introduction follow by a straightforward application of our techniques, as demonstrated in this section. To avoid tedious repetition we present only the more interesting ones here.

The general framework is as follows: First, reduce the problem of finding the best fitting shape to that of computing the smallest extent over a family of polynomials or roots of polynomials. Next, apply the following theorems to approximate the smallest extent with outliers. The details are described later in this section.

THEOREM 5.1. *Given a $(k,\varepsilon/2)$-sensitive subset $\mathcal{F}'$ of a family $\mathcal{F}$ of d-variate functions, where $0<\varepsilon<1$, one can compute*

(i) *a point $x' \in \mathbb{R}^d$ and $\delta' = \mathcal{E}_{\mathcal{F},k}(x')$ such that $\Delta^{opt}(\mathcal{F},k) \leq \delta' \leq (1+\varepsilon)\Delta^{opt}(\mathcal{F},k)$;*

(ii) *a point $\widehat{x} \in \mathbb{R}^d$ and $\widehat{\delta} = \mathcal{F}|_{\widehat{t}}^{\widehat{r}}(\widehat{x})$ such that $\widehat{r}+\widehat{t}=k$ and $\omega^* \leq \widehat{\delta} \leq (1+\varepsilon)\omega^*$, where $\omega^* = \min_{x\in\mathbb{R}^d, r+t=k} \mathcal{F}|_t^r(x)$.*

The running time is $O(n+\mathcal{C}(\mathcal{F}')^2)$, where $n=|\mathcal{F}|$ and $\mathcal{C}(\mathcal{F}')$ is the complexity of the arrangement $\mathcal{A}(\mathcal{F}')$.

Proof. We prove the first claim, as the second claim follows from a similar argument.

Each function in $\mathcal{F}$ induces a surface in $\mathbb{R}^{d+1}$. Set $x' \in \mathbb{R}^d$ to be the point such that $\mathcal{E}_{\mathcal{F}',k}(x') = \Delta^{opt}(\mathcal{F}',k) = \min_{x\in\mathbb{R}^d} \mathcal{E}_{\mathcal{F}',k}(x)$. Note that $\delta' = \mathcal{E}_{\mathcal{F},k}(x')$ can be computed in linear time once x' is specified, while x' can be obtained by computing the levels $\mathbf{L}_{\mathcal{F}',\leq k}$ and $\mathbf{U}_{\mathcal{F}',\leq k}$ and then searching, by brute force, for the shortest extent in the arrangement $\mathcal{A}(\mathcal{F}')$. It is easy to verify that this can be done in $O(n+\mathcal{C}(\mathcal{F}')^2)$ time.

Let $x^* \in \mathbb{R}^d$ be the point realizing $\mathcal{E}_{\mathcal{F},k}(x^*) = \Delta^{opt}(\mathcal{F},k)$. Since $\mathcal{F}'$ is a $(k,\varepsilon/2)$-sensitive, and $\varepsilon<1$, then by Lemma 4.3 we have

$$\begin{aligned}\Delta^{opt}(\mathcal{F},k) \leq \delta' = \mathcal{E}_{\mathcal{F},k}(x') &\leq \frac{\mathcal{E}_{\mathcal{F}',k}(x')}{1-\varepsilon/2} \\ &\leq (1+\varepsilon)\mathcal{E}_{\mathcal{F}',k}(x') \leq (1+\varepsilon)\mathcal{E}_{\mathcal{F}',k}(x^*) \\ &\leq (1+\varepsilon)\mathcal{E}_{\mathcal{F},k}(x^*) = (1+\varepsilon)\Delta^{opt}(\mathcal{F},k). \qquad \square\end{aligned}$$

Note that the above theorem implies that we not only can approximate the value of the smallest extend, but we can also find an instance that achieves the approximate value. The running time of Theorem 5.1 can be slightly improved by being more careful in the analysis, and by applying the Clarkson–Shor technique [10] to bound more tightly the number of pairs of facets of the arrangement that should be considered when computing $\Delta^{opt}(\mathcal{F}',k)$.

All the results in the remainder of this section hold with high probability, as they all rely on the randomized algorithm of Lemma 3.1.

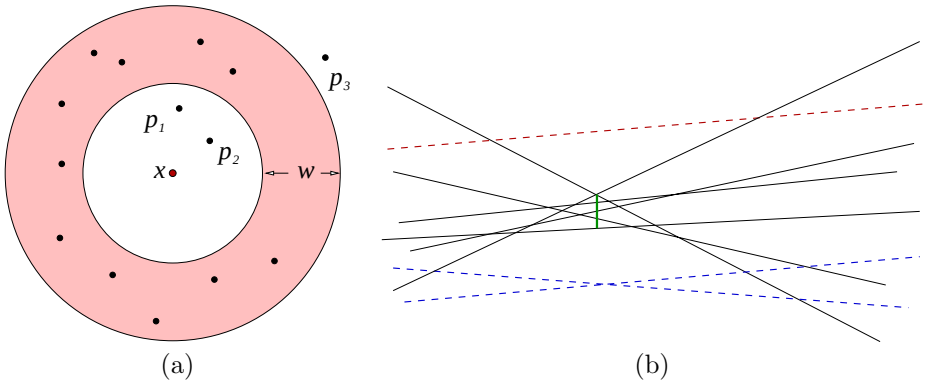


FIG. 5.1. (a) *Annulus with center x (dark region) has a width w. p_1, p_2, and p_3 are outliers.* (b) *After linearization, the thick vertical segment is the shortest extent with 3 outliers, with dotted lines corresponding to outliers p_1, p_2, and p_3.*

5.1. Minimum-width annulus/spherical shell. Given a set $P = \{p_1, \ldots, p_n\}$ of points in $\mathbb{R}^d$ and parameters $k, \varepsilon > 0$, let $\omega_{opt,k}$ denote the width of the thinnest annulus containing all but k points in P. See Figure 5.1. We would like to compute the annulus that contains at least $n - k$ points of P and whose width is at most $(1+\varepsilon)\omega_{opt,k}$. More precisely, let $d(x,p)$ denote the distance between two points $x, p \in \mathbb{R}^d$ and

$$\mu(x, P, k) = \min_{P' \subseteq P, |P'| = n-k} \left(\max_{p \in P'} d(x,p) - \min_{p \in P'} d(x,p) \right).$$

We omit k from the notation when $k = 0$. We have $\omega_{opt,k} = \min_{x \in \mathbb{R}^d} \mu(x, P, k)$, and we wish to compute a subset $P^* \subseteq P$ of size at least $n - k$ and a point $x^* \in \mathbb{R}^d$ such that $\mu(x^*, P^*) \leq (1+\varepsilon)\omega_{opt,k}$.

Let $p_i = (\xi_1, \ldots, \xi_d)$ be a point from the set P, and let $x = (x_1, \ldots, x_d)$ be an arbitrary point in $\mathbb{R}^d$. Under the Euclidean distance, we define

$$f_i(x) = d(p_i, x) = \sqrt{\sum_{1 \leq j \leq d} x_j^2 - 2 \sum_{1 \leq j \leq d} \xi_j x_j + \sum_{1 \leq j \leq d} \xi_j^2}.$$

Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ denote the set of functions defined by the points of P.

Observe that $\omega_{opt,k}$ is in fact the shortest (r,t)-extent for $\mathcal{F}$ such that $r + t = k$. By Theorem 4.4, we can compute, in $O(n + k/\varepsilon^{2d})$ time, a subset $\mathcal{F}' \subseteq \mathcal{F}$ which is a (k, ε)-sensitive for $\mathcal{F}$ and $|\mathcal{F}'| = O(k/\varepsilon^{2d})$. Since the image of each function of $\mathcal{F}$ is a cone in $\mathbb{R}^{d+1}$, one can thus approximate $\omega_{opt,k}$ in

$$O\left(n + |\mathcal{F}'|^{2(d+1)}\right) = O\left(n + \frac{k^{2(d+1)}}{\varepsilon^{4d(d+1)}}\right)$$

time using the algorithm of Theorem 5.1.

THEOREM 5.2. *Given a set P of n points and parameters $k, \varepsilon > 0$, one can compute an $x^* \in \mathbb{R}^d$, and $P^* \subseteq P$, where $|P^*| \geq n - k$ such that $\mu(x^*, P^*) \leq (1+\varepsilon)\omega_{opt,k}$ in $O(n + k^{2d+1}/\varepsilon^{4d(d+1)})$ time.*

5.2. Minimum-volume annulus/spherical shell. Let $\mathcal{B}(x, R, r)$ denote the shell between two concentric balls, centered at point x, with radii R and r, respectively.

That is, $\mathcal{B}(x, R, r)$ is composed of all points inside the outer ball and outside the inner ball. Given a set of points $P \in \mathbb{R}^d$, we would like to find a spherical shell whose volume is minimized, and which contains at least $n - k$ points of P, over all $x \in \mathbb{R}^d$ and $R, r \in \mathbb{R}$. Let $\mathcal{SS}_{opt,k}(P)$ denote the volume of this spherical shell. This problem is easier than the minimum-width variant, as it can be reduced to linear programming (this is folklore; it is also described in [1]).

The current fastest algorithm for the exact problem, in two dimensions, with k outliers is due to Chan [7], and it works in $O\big(n \log k + k^{11/4} n^{1/4} \log^c k\big)$ time.

THEOREM 5.3. *Given a set P of n points and parameters $k, \varepsilon > 0$, one can compute a set P such that* (i) $P' \subseteq P$, (ii) P *is a (k, ε)-coreset for the minimum area spherical shell measure, and* (iii) $|P'| = O(k/\varepsilon^d)$.

In particular, one can compute a spherical shell whose volume ε-approximates $\mathcal{SS}_{opt,k}(P)$ in $O(n+k^{2(d+1)}/\varepsilon^{2d(d+1)})$ time for $d > 2$ and in $O(n+\frac{k}{\varepsilon^2}\log\frac{k}{\varepsilon}+\frac{k^3}{\varepsilon^{1/2}}\log^c k)$ time for $d = 2$, where c is a constant.

Proof. Using linearization, computing $\mathcal{SS}_{opt}(P)$ is reduced to computing the extent of n hyperplanes in $\mathbb{R}^{d+1}$ [4]. Thus, by Theorem 3.13, there is a subset P' of P, which is (k, ε)-sensitive for the measure $\mathcal{SS}_{opt}(\cdot)$, and $|P'| = O(k/\varepsilon^d)$.

For $d > 2$, the running time is just the result of applying Theorem 5.1. As for $d = 2$, we use the algorithm of Chan [7] on the subset P'. ∎

5.3. Minimum-width cylindrical shell. Given a line ℓ in $\mathbb{R}^d$ and two real numbers $0 \le r \le R$, the cylindrical shell $S(\ell, r, R)$ is the closed region lying between the two coaxial cylinders of radii r and R, respectively, with ℓ as their axis. The width of shell $S(\ell, r, R)$ is $R - r$. Given a set of n points P, let $\mathcal{CS}^*$ denote the minimum width among all cylindrical shells that enclose at least $n - k$ points of P. In the *approximate minimum-width cylindrical shell with outliers* problem, we would like to compute a cylindrical shell containing at least $n - k$ points from P whose width is at most $(1 + \varepsilon)\mathcal{CS}^*$.

THEOREM 5.4. *Given a set P of n points in $\mathbb{R}^d$ and parameters $k, \varepsilon > 0$, one can compute in $O(n + k^{cd^2}/\varepsilon^{c'd^4})$ time a line ℓ^* and $P^* \subseteq P$, where $|P^*| \ge n - k$, such that the minimum width of the cylindrical shell containing P^* with axis ℓ^* is smaller than $(1 + \varepsilon)\mathcal{CS}^*$. Here c, c' are constants independent of d, ε, and n.*

Proof. Using linearization, computing $\mathcal{CS}^*$ is reduced to computing the extent of n hyperplanes in $\mathbb{R}^{O(d^2)}$ [4]. Thus, by Theorem 3.13, there is a subset P' of P, which is (k, ε)-sensitive for the width measure, and $|P'| = O(k/\varepsilon^{O(d^2)})$. The running time is just the result of applying Theorem 5.1 to this set. ∎

5.4. Moving points. The same technique can also be extended to approximate various measures of moving points. More precisely, given a set P of n points moving in $\mathbb{R}^d$, for each $p \in P$, let $p(t) = (\xi_1(t), \ldots, \xi_d(t))$ denote the position of point p at time t, and let $P(t)$ denote the set P at time t. We say that the motion of P is *algebraic* of degree ν if the functions $\xi_1(t), \ldots, \xi_d(t)$ are polynomials of degree at most ν for all the points of P. For simplicity, we assume in the remaining part that $\nu = 1$; i.e., each point moves along a straight line.

Let $\mu_k(P)$ denote some measure of the point set P, such as width (with at most k outliers). For any $\varepsilon > 0$, we say that a subset $Q \subseteq P$ ε-approximates P with respect to μ_k if at any time t, $(1 - \varepsilon)\mu_k(P(t)) \le \mu_k(Q(t)) \le \mu_k(P(t))$. Arguing as in [4], we get the following result.

THEOREM 5.5. *Given a point-set P with n linearly moving points and parameters $k, \varepsilon > 0$, one can compute, in $O(n + k/\varepsilon^c)$ time, a subset $Q \subseteq P$ of size $O(k/\varepsilon^c)$ such*

that Q is an ε-approximation to P under measures with k outliers such as diameter, width, minimum-radius enclosing ball, projection width, minimum-width annulus, and minimum-width cylindrical shell. Here c is a constant that depends on the dimension d and the measure considered.

5.5. Handling updates—insertions and deletions. By building a balanced binary tree over the given point set and maintaining in each node the coreset of all the points in its subtree, one can maintain a coreset under insertions and deletions, where every insertion and deletion is handled in polylogarithmic time. The following theorem follows in a plug-and-play fashion by combining the techniques of [3] with our techniques.

THEOREM 5.6. *Let P be a set of points in $\mathbb{R}^d$, let k and $\varepsilon > 0$ be parameters, and let $\mu(\cdot)$ be one of the measures discussed before. Suppose that for any $Q \subseteq P$, we can compute a (k, ε)-sensitive (for $\mu(\cdot)$) subset S of Q of size $O(k/\varepsilon^\rho)$ in $O(|Q| + f(\varepsilon))$ time. Then we can maintain a (k, ε)-sensitive subset of P of size $O(k/\varepsilon^\rho)$ under insertion/deletion operations in $O(((\log n)/\varepsilon)^\rho + f(\varepsilon/\log n)\log n)$ time per update.*

Note that Theorem 5.6 implies that we can maintain the approximate optimal solution for *all* of the measures discussed above, under insertions and deletions, with k outliers. The time to handle an update is $O(\mathrm{poly}(k, 1/\varepsilon, \log n))$, where $\mathrm{poly}(\cdots)$ is a constant degree polynomial in its parameters, where the constant is a function of d and the measure being approximated.

6. Conclusions. We presented a general approximation algorithm for shape fitting that can handle outliers efficiently with near linear running time when the number of outliers is small. The main contribution of this paper is the proof that there exists a small coreset for various shape fitting problems when considering outliers. The authors believe that the existence of such a small coreset is quite surprising.

6.1. Comparison to linear programming with violations. It is beneficial to compare our result to the algorithms known for linear programming with violations. Mulmuley [23] showed, using the Clarkson–Shor technique, that in an arrangement of n hyperplanes in d dimensions, there are at most $O(k^d)$ *local* minima in the top k-level. The argument relies on picking a random sample of n/k hyperplanes and showing that the probability that a local minima in the top k-level is the minimum of the sample (i.e., the lowest point on the upper envelope of the sample of hyperplanes) is $\Omega(1/k^d)$, and since there is only one local minima in the upper envelope of the sample, it follows that the global number of local minima in the first top k-levels is $O(k^d)$. Matoušek [22] solves the linear programming with violations by generating all those local minima explicitly (in amortized $O(n)$ time for each one by solving instances of linear programming) and returning the best one.

A straightforward adaption of this argument shows that there are $O(k^{d+1})$ minima to the k-extent function $\mathcal{E}_{\mathcal{H},k}$. In our setting, since we are interested only in the extent in a subspace $X \subseteq \mathbb{R}^{d-1}$, it is no longer true that such a bound (that depends only on k) holds on the number of minima of the extent when restricted to X. Specifically, it is easy to find examples where the number of local minima of the k-extent is $\Omega(n)$ when restricted to X. We briefly describe such an example below. Imagine a set of functions $\mathcal{F}$ where each $f \in \mathcal{F}$ is a parabola $f(x) = ax^2 + bx + c$ (see Figure 6.1). The number of local minima of the 0-extent is $\Omega(n)$, as indicated by the dotted vertical segments in Figure 6.1. Now we linearize each f into a plane $h(x, y) = ay + bx + c$ in $\mathbb{R}^3$, with the linearization image being $\eta(x) = (x, x^2)$; call the resulting set of planes $\mathcal{H}$. The set X, in this case, is the image of the linearization, namely, $X = \eta(\mathbb{R}) = \{(x, x^2) \mid x \in \mathbb{R}\}$.

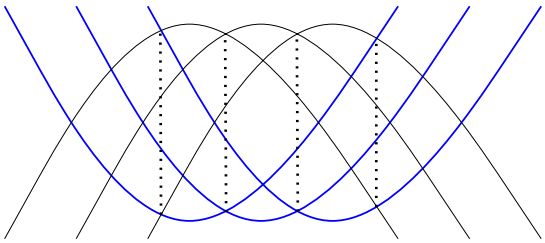


FIG. 6.1. *The graphs of a set of functions $\mathcal{F}$, with each being a parabola. Dotted vertical segments depict the local minima of* 0*-extent in the arrangement of $\mathcal{F}$.*

Observe, that the number of local minima of the 0-extent along $\eta(\mathbb{R})$ for $\mathcal{H}$ remains the same. Therefore, while the number of minima of the 0-extent with respect to $\mathbb{R}^2$ is $O(1)$ (i.e., there is one minimum in the upper and lower envelop of the arrangement of $\mathcal{H}$), the number of minima when restricted to X is $\Omega(n)$. Furthermore, the problem of finding the shortest vertical segment when restricted to X (even without outliers) is no longer an LP-type problem, and as such we can no longer move efficiently from one minimum to another using Matoušek's approach.

Nevertheless, this gives a new interpretation of our result. It implies that if we are interested only in approximation, the number of approximate local minima is bounded by a polynomial in k (while the exact one is *not*). Furthermore, our coreset result yields a direct way to generate those local minima.

6.2. General discussion. A limitation of our technique is that the number of outliers has to be moderately small (i.e., $O(n^{1/2d})$) to achieve near linear running time. Erickson and Seidel [17, 16] show that to determine whether a set of n points in $\mathbb{R}^d$ contains $d+1$ points on a common hyperplane requires $\Omega(n^d)$ sidedness queries. (Given $d+1$ points $p_0, \ldots, p_d$, the *sidedness query* decides on which side of the oriented hyperplane defined by $p_1, \ldots, p_n$ the point p_0 lies.) This lower bound holds in a decision tree model of computation in which every decision is based on the result of a sidedness query, which they argue is a reasonable model. Their result implies that when the number of outliers is huge (i.e., $k = n - d - 1$), a near-linear time approximation algorithm cannot be achieved. We leave the problem of improving the running time of our algorithm as an open question for further research.

This raises the question whether one can find "universal" approximation algorithms with outliers—namely, approximation algorithms that have near-linear running time independent of the number of outliers. Currently, the only case the authors are aware of that such an algorithm exists is for the case of minimum enclosing circles with k outliers; see [21] for the currently fastest algorithm known for this problem.

Of course, using standard ε-sampling techniques, one can achieve *combinatorial* approximation of k levels for relatively large k efficiently. However, this provides a different and *weaker* type of approximation than the one presented in the paper. In particular, our results imply that we can compute an approximation when the number of outliers is fixed, while the geometric measure is approximated. Using ε-sampling implies a result where the number of outliers is approximated. This is acceptable only when the number of outliers is relatively large.

Another limitation of our technique (which is inherent in almost all the work concerning outliers [13]) is that the number of outliers, k, has to be specified in the input. In practice, there are situations when this information is not known beforehand.

How to formulate a reasonable problem without k being the input, and then solving it, are among some of the future directions of research.

Finally, note that, for $k = 0$, this paper provides an alternative construction to the techniques of [4]. The approximation techniques presented here are simpler to implement, but the resulting coresets (not surprisingly) are larger.

Acknowledgment. The authors would like to thank the anonymous referees for their useful comments.

REFERENCES

[1] P. K. Agarwal, B. Aronov, S. Har-Peled, and M. Sharir, *Approximation and exact algorithms for minimum-width annuli and shells*, Discrete Comput. Geom., 24 (2000), pp. 687–705.

[2] P. K. Agarwal, A. Efrat, and M. Sharir, *Vertical decomposition of shallow levels in* 3*-dimensional arrangements and its applications*, SIAM J. Comput., 29 (1999), pp. 912–953.

[3] P. K. Agarwal and S. Har-Peled, *Maintaining approximate extent measures of moving points*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 148–157.

[4] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, *Approximating extent measures of points*, J. ACM, to appear.

[5] P. K. Agarwal and J. Matoušek, *On range searching with semialgebraic sets*, Discrete Comput. Geom., 11 (1994), pp. 393–418.

[6] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed., Wiley-Interscience, New York, 2000.

[7] T. M. Chan, *Low-dimensional linear programming with violations*, in Proceedings of the 43th Annual IEEE Symposium Found. Comput. Sci., IEEE Computer Society Press, Los Alamitos, CA, 2002, pp. 570–579.

[8] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan, *Algorithms for facility location problems with outliers*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 642–651.

[9] B. Chazelle, *The Discrepancy Method*, Cambridge University Press, Cambridge, UK, 2000.

[10] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry,* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[11] N. Cristianini and J. Shaw-Taylor, *Support Vector Machines*, Cambridge University Press, Cambridge, UK, 2000.

[12] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid, *Static and dynamic algorithms for* k*-point clustering problems*, J. Algorithms, 19 (1995), pp. 474–503.

[13] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed., Wiley-Interscience, New York, 2001.

[14] J. Dunagan and S. Vempala, *Optimal outlier removal in high-dimensional spaces*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2001, pp. 627–636.

[15] D. Eppstein and J. Erickson, *Iterated nearest neighbors and finding minimal polytopes*, Discrete Comput. Geom., 11 (1994), pp. 321–350.

[16] J. Erickson, *New lower bounds for convex hull problems in odd dimensions*, SIAM J. Comput., 28 (1999), pp. 1198–1214.

[17] J. Erickson and R. Seidel, *Better lower bounds on detecting affine and spherical degeneracies*, Discrete Comput. Geom., 13 (1995), pp. 41–57.

[18] J. Gao and L. J. Guibas, *Staying in the middle: Approximate medians in* $\mathbb{R}^1$ *and* $\mathbb{R}^2$ *for moving points*.

[19] S. Har-Peled and K. R. Varadarajan, *Approximate shape fitting via linearization*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 66–73.

[20] S. Har-Peled and Y. Wang, *Shape fitting with outliers*, www.uiuc.edu/~sariel/research/papers/02/outliers/, 2002.

[21] J. Matoušek, *On enclosing* k *points by a circle*, Inform. Process. Lett., 53 (1995), pp. 217–221.

[22] J. Matoušek, *On geometric optimization with few violated constraints*, Discrete Comput. Geom., 14 (1995), pp. 365–384.

[23] K. Mulmuley, *On levels in arrangements and Voronoi diagrams*, Discrete Comput. Geom., 6 (1991), pp. 307–338.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 286–303

NEW STABILITY RESULTS FOR ADVERSARIAL QUEUING*

ZVI LOTKER†, BOAZ PATT-SHAMIR†, AND ADI ROSÉN‡

Abstract. We consider the model of "adversarial queuing theory" for packet networks introduced by Borodin et al. [*J. ACM*, 48 (2001), pp. 13–38]. We show that the scheduling protocol first-in-first-out (FIFO) can be unstable at any injection rate larger than $1/2$ and that it is always stable if the injection rate is less than $1/d$, where d is the length of the longest route used by any packet. We further show that *every* work-conserving (i.e., greedy) scheduling policy is stable if the injection rate is less than $1/(d+1)$.

Key words. adversarial queuing theory, network protocols, stability, lower bounds

AMS subject classifications. 68M12, 68M20, 68Q25, 68W15

DOI. 10.1137/S0097539702413306

1. Introduction.

Recent years have seen a growing amount of work being concentrated on analyzing packet networks under nonprobabilistic scenarios rather than under probabilistic assumptions (see, e.g., [7, 4, 1, 14, 12, 13, 3, 5]). Much of this work makes use of the model of "adversarial queuing theory" proposed by Borodin et al. [7]. The model can be briefly described as follows. Time proceeds in discrete steps. In each step, packets are injected into the network with their routes. Each packet traverses its respective route hop by hop in a store-and-forward fashion. In each time step, one packet may cross each link, and all other packets waiting for that link are stored in a buffer at the tail of that link. The behavior of the system is determined by the *queuing policy*. The queuing policy chooses, at each time step, for each link, which of the competing packets should be forwarded over that link. One of the main questions in the adversarial queuing model is the question of *stability*. That is, under what conditions is there a bound on the size of the link buffers, as opposed to them growing to infinity as time proceeds? The conditions involve the topology of the network, the queuing policy used, and the injection pattern of the packets. The latter is characterized in the framework of adversarial queuing theory by the *rate* at which packets are injected. Intuitively, the rate of injection is said to be r if, for every link e in the network, the average number of packets requiring e, injected by the adversary in any time step, is at most r (a formal definition of the model is given in section 2). Note that in this model one does not assume any probabilistic assumptions on the behavior of the traffic. Rather, answers are sought under the only assumption that the total bandwidth requested by the adversary is not more than the total bandwidth the network provides.

In the framework of adversarial queuing theory, it is known that some networks are stable for *every* greedy protocol as long as the rate of injection is less than 1, while other networks do not exhibit this phenomenon [7, 4]. The networks which

*Received by the editors August 21, 2002; accepted for publication (in revised form) June 17, 2003; published electronically January 22, 2004. A preliminary version of this paper appeared in *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, Winnepeg, MB, Canada, 2002, pp. 192–199.

http://www.siam.org/journals/sicomp/33-2/41330.html

†Department of Electrical Engineering, Tel Aviv University, Tel Aviv, 69978, Israel (zvilo@eng.tau.ac.il, boaz@eng.tau.ac.il).

‡Department of Computer Science, Technion, Haifa, 32000, Israel (adiro@technion.ac.il). This author's research was supported in part by the Fund for the Promotion of Research at the Technion and by the Technion V.P.R. Fund.

are always stable have been named "universally stable" networks [7] and have been fully characterized [14, 2]. From the point of view of protocols, some protocols are known to be universally stable; i.e., they are stable on any network topology for any rate of injection $r < 1$. Such protocols are, for example, longest-in-system (LIS) and furthest-to-go (FTG). Other natural protocols, however, are known not to always be stable, e.g., first-in-first-out (FIFO), nearest-to-go (NTG), last-in-first-out (LIFO), and furthest-from-source (FFS) [4]. Furthermore, the protocol NTG (and FFS and LIFO) exhibit the phenomenon of being unstable on certain networks even at arbitrarily low injection rates [7]. Previous papers in this area mentioned as one of the main interesting open problems the question of determining the rate at which the (very commonly used) FIFO policy is guaranteed to be stable and if such rate exists at all. Prior to the present work, it was known that FIFO is not universally stable and that it can be unstable for $r > 0.85$ [4]. This bound was improved to 0.8357 by Díaz et al. [11] and further improved to 0.749 by Koukopoulos, Nikoletseas, and Spirakis [15]. Díaz et al. [11] also presented a formula to calculate, for any given network, a bound so that FIFO is stable on that network if the injection rate is below that bound. In particular, they consider as parameters the number of edges in the network, denoted m, the length of the longest route used by any packet, denoted d, and the maximum in-degree in the network, denoted α; their bound is at most $\frac{1}{2dm\alpha}$ for any network.

The contribution of this paper is twofold. First, we show that FIFO can be unstable for any rate greater than $1/2$ and that, on the other hand, FIFO is always stable if the rate is less than $1/d$. Second, we extend the stability proof for FIFO to show that *any* greedy policy is stable if the injection rate is less than $1/(d+1)$, while previously it was only known for general greedy protocols that the system is stable if the injection rate is bounded by $1/m$ [6].[1] We remark that our stability proofs do not only show that the buffers have bounded size if the rate is sufficiently low. They show, in addition, that the buffer size in this case has an upper bound independent of network parameters (depending only on the parameters of the adversary).

Our instability proof entails new techniques that greatly simplify the analysis of the FIFO policy. In particular, we develop a technique that enables us to construct adversaries for some *acyclic* parameterized networks that we call "gadgets" and then compose these gadgets and adversaries to form a cyclic network and a single adversary that together show instability. Furthermore, we simplify the specification of the adversary by defining conditions under which the adversary is allowed to "reroute" packets, thus allowing us to specify routes for the packets "on the fly."

Recently, subsequent to the initial publication of the present work [17], Bhattacharjee and Goel proved that FIFO can be unstable at arbitrarily low injection rates [8]. Some of the techniques used in their work (such as concatenation of parameterized gadgets and packet rerouting) are similar to the techniques we use in the present work.

Organization. The rest of this paper is organized as follows. In section 2 we define the model formally. In section 3 we prove that FIFO can be unstable for any rate greater than $1/2$. In section 4 we prove our stability results. We conclude with some remarks in section 5.

2. Formal model. We use the adversarial queuing model [7], defined as follows. The communication network is modeled by a directed graph $G = (V, E)$, and we

[1] Recently, we learned that Zhang, Duan, and Hou [18] proved that FIFO is stable for injection rates less than $1/d$. This result was obtained independently of ours.

denote $|V| = n$, $|E| = m$. Each node $v \in V$ represents a communication switch, and each edge $e \in E$ represents a link between two switches. In each node, there is a *buffer* associated with each outgoing link. Buffers store *packets*. Packets are *injected* into the network with a *route*, which is a simple directed path in G. When a packet is injected, it is placed in the buffer of the first link on its route. The system proceeds in global time steps numbered $0, 1, \ldots$. Each time step is divided into two substeps. In the first substep, one packet is sent from each nonempty buffer over its corresponding link. In the second substep, packets are received by the nodes at the other end of the links; they are absorbed (eliminated) if that node is their destination, and otherwise they are placed in the buffer of the next link on their respective routes. In addition, new packets are injected in the second substep.

The task of the *protocol* is to select which packet to send over a link if there is more than one packet in the buffer associated with that link. We remark that we are interested in *greedy* protocols (in fact, the definitions above allow only such protocols), in which a link cannot be idle in a time step if its buffer is nonempty in the first substep. The protocol FIFO selects the packets to be sent from a buffer in the same order as their arrival order at that buffer.

The injection of the packets into the network is modeled as being done by an *adversary*. Following [7], we use the following parameterized definition for the adversary.

DEFINITION 2.1. *Let $\mathcal{A}$ be an adversary. $\mathcal{A}$ is called a (w, r) adversary if, for some $r \leq 1$, called the* rate *of $\mathcal{A}$, and some integer $w \geq 1$, called the* window size *of $\mathcal{A}$, the following holds. For any time $t \in \mathcal{N}$, let $\mathcal{I}^t$ be the set of packets injected during the w time steps from t to $t + w - 1$, inclusive. Let Π^t be the set of paths that the packets in $\mathcal{I}^t$ have to follow. Then the maximum number of times any edge appears in Π^t is at most rw.*

For our instability results we use a weaker adversary, which is not allowed to inject bursty traffic. We call this adversary a *rate-r adversary* [4]: for every interval of time of length t and every edge e, a rate-r adversary may inject at most $\lceil rt \rceil$ packets whose routes require e.

3. Instability of FIFO. In this section we prove that FIFO can be unstable at rate $\frac{1}{2} + \epsilon$ for any $\epsilon > 0$. The high level view of the proof is as follows. First, we define a small acyclic graph called "gadget," which has special "ingress" and "egress" edges. Gadgets can be composed in series by identifying the egress edge of one gadget with the ingress edge of its successor, getting a "daisy chain." We show that a rate-r adversary (for $r > \frac{1}{2}$) can increase the size of a given queue in the ingress edge of the chain by any desired factor to get a large queue at the egress edge of the chain (using a sufficiently long chain). We then prove that a queue in the egress edge of the chain can be translated to a queue of fresh packets in the ingress edge of the chain by losing only a fraction of the size of the queue.

Since in our construction packets have long routes, we find it more convenient to specify the routes in an "on-line" fashion. That is, when we construct the adversary, we do not specify the complete routes of the packets when they are injected (even though we can, in principle). Rather, we prove below some conditions that allow us to reroute packets without violating the capacity constraints. Formally, this is done by altering the adversary. We find this technique useful in the sense that it makes the construction more "localized." We stress that this is just a matter of representation: the actual adversary used to prove the results is the same rate-r adversary used, e.g., in [4, 11, 15].

The proof is structured as follows. In section 3.1 we specify the conditions under which packets can be rerouted. In section 3.2 we specify and analyze a rate-r adversary for two daisy-chained gadgets. Some small adversaries used for "gluing" and the overall adversary are specified in section 3.3.

3.1. Packet rerouting. In this section we prove a technical lemma that allows us to construct adversaries "on the fly" for FIFO. Informally, it says that if there is a set of packets that have routes that already share a single edge, then these packets can be arbitrarily rerouted as long as they are routed to new edges. In fact, the rerouting technique can be applied to a large class of queuing policies defined below.

DEFINITION 3.1. *A queue policy is called* historic *if the scheduling decisions are independent of the remaining routes beyond the next edge of each packet.*

Note that policies that are based on the arrival time at the buffer (such as FIFO and LIFO), on the injection time (e.g., LIS and NIS), or on the route from the source (e.g., FFS) are examples of historic policies. Note that a historic queue policy must not even depend on the destinations of the packets. For example, FTG and NTG are not historic. (Historic policies are called *nonpredictive* in [16].)

First, we define formally the notion of new edges.

DEFINITION 3.2. *Let G be a graph, $\mathcal{Q}$ a queuing policy, and $\mathcal{A}$ a rate-r adversary. Let t be a time step in the execution of $\mathcal{Q}$ in G under $\mathcal{A}$. Let P be a subset of the packets that are in the network at time t. Let t^* be the minimum injection time of all packets in P. An edge e is* new to P *if e is not a member of any route of a packet (either in P or not) injected by $\mathcal{A}$ at times $\tau \geq t^* - \lceil \frac{1}{r} \rceil$.*

We remark that since in this paper we deal with rates larger than $\frac{1}{2}$, then $\lceil \frac{1}{r} \rceil \leq 2$.

We can now state and prove our rerouting claim.

LEMMA 3.3. *Let $\mathcal{Q}$ be a deterministic historic queue policy, G a graph, $\mathcal{A}$ a rate-r adversary, and t a time step. Let $P(t)$ be the set of packets in the network at time t. For each $p \in P(t)$, denote the next edge to be traversed by p at time t by e_p, and denote the complete path of p by $q_p e_p r_p$. Let $P_0 \subseteq P(t)$ be a set of packets whose routes have at least one edge common to all. Then for any set of paths $\{r'_p \mid p \in P_0\}$ that consist of edges that are new to $P(t)$, there exists a rate-r adversary $\mathcal{A}'$ such that the following holds true.*

1. *The execution of the system under $\mathcal{A}$ and $\mathcal{A}'$ is identical until time t.*
2. *For every packet p injected by $\mathcal{A}$ there is a packet p injected by $\mathcal{A}'$ at the same time.*
3. *If $p \in P_0$, then its route under $\mathcal{A}'$ is $q_p e_p r'_p$.*
4. *If $p \notin P_0$, then its route under $\mathcal{A}'$ is $q_p e_p r_p$.*

Proof. Define $\mathcal{A}'$ as follows. $\mathcal{A}'$ injects the same number of packets as $\mathcal{A}$ and at the same times. For $p \in P_0$, set the route of p to $q_p e_p r'_p$. For $p \notin P_0$ set the route of p to $q_p e_p r_p$. Clearly, claims (1), (2), (3), and (4) follow directly from the assumption that $\mathcal{Q}$ is historic and by the construction. We need only to verify that $\mathcal{A}'$ is a rate-r adversary. To see this, first note that the load on any non-new edge may have only been reduced. Now, consider any edge e in $\bigcup_{p \in P_0} r'_p$. Let $\hat{e}$ be the edge common to the routes of all $p \in P_0$. Let t^* be the minimum injection time over all packets in $P(t)$.

Consider any time interval $[t_1, t_2]$. If $t_2 < t^*$, then the number of packets injected in $[t_1, t_2]$ by $\mathcal{A}'$ and that require e is equal to the number of packets injected in $[t_1, t_2]$ by $\mathcal{A}$ and that require e. If $t_2 \geq t^*$ we consider time intervals $I = [t_1, t^*)$ and $I' = [t^*, t_2]$. For interval I the number of packets injected by $\mathcal{A}'$ and that require e is the same as for $\mathcal{A}'$, which is at most $\lceil ((t^* - \lceil \frac{1}{r} \rceil) - t_1) r \rceil$, since e is a new edge with

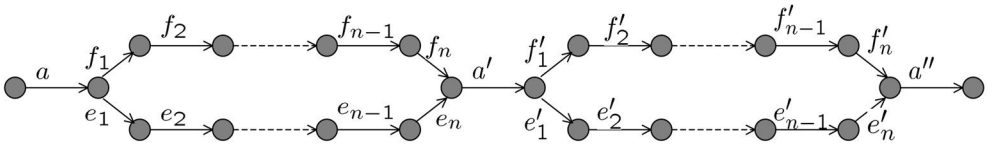


FIG. 3.1. *The graph F_n^2: two F_n gadgets glued together. The left gadget is called F, and the right gadget is called F'. Edge a' is the egress of F and the ingress of F'.*

respect to $P_0 \subseteq P(t)$ and time t^* is the minimum injection time of all packets in $P(t)$ (see Definition 3.2). For interval I' the number of packets injected in I' by $\mathcal{A}'$ and that require e is at most the number of packets injected in I' by $\mathcal{A}$ and that require $\hat{e}$. This is at most $\lceil (t_2 - t^* + 1)r \rceil$. The total number of packets injected by $\mathcal{A}'$ in $[t_1, t_2]$ and that require e is therefore at most $\lceil ((t^* - \lceil \frac{1}{r} \rceil) - t_1)r \rceil + \lceil (t_2 - t^* + 1)r \rceil$. This is at most $\lceil (t_2 - t_1 + 1)r \rceil$, as required. □

Remark 1. Lemma 3.3 allows us to use a "dynamic" adversary that changes the routes of packets on-line. However, this is only a matter of presentation: we do not change the power of the adversary; we only construct it in a succession of refinements. The main advantage of the lemma is that it allows us to modify the remainder of the routes arbitrarily, under the specified restrictions (shared edge in old routes, new edges in modified routes, and historic policy); we do not have to worry about capacity constraints of new edges.

Remark 2. Note that a packet may be rerouted several times, as long as the number of reroutings is finite.

3.2. Gadgets and their adversaries. We now define the gadgets that we use and their local adversaries.

DEFINITION 3.4. *A* gadget *is a directed acyclic graph with one edge called* ingress *emanating from a degree-*1 *source, and one edge called* egress*, leading to a degree-*1 *sink. Given two gadgets G, H, define $G \circ H$ to be the gadget that results from identifying the egress of G with the ingress of H. The ingress of $G \circ H$ is the ingress of G, and the egress of $G \circ H$ is the egress of H.*

For any gadget F, let F^0 denote the single edge graph which is both ingress and egress. For $i > 0$, we denote $F^i = F^{i-1} \circ F$. We call the "$\circ$" operation *daisy-chaining*.

We will use a parametric gadget denoted F_n, which consists of ingress edge a, egress edge a', and two parallel paths of length n from the ingress edge to the egress edge, whose edges are denoted $e_1, \ldots, e_n$ and $f_1, \ldots, f_n$. Figure 3.1 shows F_n^2.

We will construct an adversary that maintains the following *gadget invariant*.

DEFINITION 3.5. *$C(S, F_n)$ is said to be true at a given time if the following holds at that time on graph F_n.*

1. *The total number of packets in the buffers of $e_1, \ldots, e_n$ is S.*
2. *For each $i = 1, \ldots, n$, the buffer of e_i is nonempty, and the packets in e_i have remaining routes $e_i, e_{i+1}, \ldots, e_n, a'$.*
3. *There are S packets in the buffer of edge a, all with the same remaining route $a, f_1, \ldots, f_n, a'$.*
4. *There are no other packets in F_n.*

In our construction, we use a daisy chain of many gadgets. However, we start by considering two daisy-chained gadgets, namely the graph F_n^2 (Figure 3.1). We denote the first gadget of F_n^2 by F, and the second by F', and add a prime to the name of all edges in F'. The conditions of the following lemma are designed to allow repeated

rerouting, but essentially the idea is to have the condition $C(S,F)$ carry over from one gadget to the next, with a larger value S.

LEMMA 3.6. *Let $r = \frac{1}{2} + \epsilon$ for some $\epsilon > 0$. There exist numbers n and S_0 that depend on ϵ, such that for any $S > S_0$, if in the graph F_n^2 we have that for some time τ, all packets present at time τ were injected after time τ_0, and*

- *$C(S,F)$ holds at time τ, and*
- *F' is empty at time τ, and*
- *no packets using edges in F' were injected in the time interval $[\tau_0 - \lceil 1/r \rceil, \tau]$,*

then there exists a rate-r adversary for F_n^2 such that at time $\tau + 2S + n$, $C(S', F')$ holds for some $S' \geq S(1+\epsilon)$, and F is empty.

Proof. To define the adversary, we use the notation $R_i \stackrel{\text{def}}{=} \frac{1-r}{1-r^i}$ for $1 \leq i \leq n$. Note, for later reference, that for all i,

$$\frac{R_i}{r + R_i} = R_{i+1} \ . \tag{3.1}$$

We first choose parameters under the constraints below:

$$n > \max\left(\frac{\log\epsilon - 2}{\log r}, \quad 1 - \frac{1}{\log r}\right),$$
$$S_0 > \max\left(2n, \quad \frac{n}{2(R_n - R_{n+1})}\right).$$

We remark that for small ϵ values, we get $n = \Theta(\log \frac{1}{\epsilon})$ and $S_0 = \Theta(nr^{-n}) = \Theta(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ (see the appendix for a detailed derivation of the asymptotic bounds).

Let us assume, for simplicity of notation, that $\tau = 0$. We now specify the adversary that will create a situation where $C(S', F')$ holds for $S' = 2S(1 - R_n)$. In the adversary specification, as well as in the ensuing analysis, we ignore floors and ceilings for the sake of simplicity of presentation. We remark that carrying these throughout the computations would add only additive terms that can be compensated for by using a larger S_0 value (cf. [4, 11, 15]).

The adversary is as follows.

(1) Extend the routes of all packets stored in F at time 0 by adding the path $e'_1, \ldots, e'_n, a''$.
(2) For every edge e'_i in F' ($i = 1, \ldots, n$), packets are injected at rate r in the time steps $i, i+1, \ldots, i+t_i$, where $t_i \stackrel{\text{def}}{=} \frac{2S}{r+R_i}$. The route of each of these packets is the single edge e'_i.
(3) In the time interval $[1, S]$, rS packets are injected, at rate r, with route $a, f_1, \ldots, f_n, a', f'_1, \ldots, f'_n, a''$.
(4) Let $X = S' - rS + n$. X packets are injected in the first $X \cdot \frac{1}{r}$ time steps of the interval $[S+n+1, S+n+S]$, with routes $a', f'_1, \ldots, f'_n, a''$. (We show later that $0 \leq X \leq rS$.)

First, note that this is a rate-r adversary: part (1) is justified by Lemma 3.3, since the routes of all packets stored in F share the edge a', and the extensions are for new edges as defined in Definition 3.2.

Edges $e'_1, \ldots, e'_n$ are used only by part (2) at rate r. Edges $f'_1, \ldots, f'_n$ and a'' are used at rate r in parts (3) and (4), which cover disjoint time intervals. It remains to show that $0 \leq X \leq rS$.

CLAIM 3.7. *For every $r < 1$, we have $0 < X \leq rS$.*

Proof. First we prove that $X > 0$. By definitions,

$$\begin{aligned} X \;>\; X - n &= S' - rS \\ &= 2S(1 - R_n) - rS \\ &= S\left(2 - \frac{2-2r}{1-r^n} - r\right) . \end{aligned}$$

Now,

$$\begin{aligned} 2 - \frac{2-2r}{1-r^n} - r &= \frac{r - 2r^n + r^{n+1}}{1-r^n} \\ &> \frac{r-2r^n}{1-r^n} \\ &> 2r(1-2r^{n-1}) \\ &> 0 , \end{aligned}$$

since $r^n < r^{n-1} < 1/2$ by the choice of n, and hence $X > 0$. Next we prove that $X \le rS$. By the definitions,

$$\begin{aligned} rS - X &= rS - (2S(1-R_n) - rS + n) \\ &= 2S(r + R_n - 1) - n . \end{aligned}$$

Since $S \ge S_0 > \frac{n}{2(R_n - R_{n+1})} \ge \frac{n}{2(r+R_n-1)}$ by assumption, we get $rS - X > 0$. □

We now show that in fact at time $2S + n$, $C(S', F')$ holds and that F is empty. This will be sufficient, since by the definition of S' we have that

$$\begin{aligned} S' &= 2S(1-R_n) \\ &= 2S\left(\frac{r}{1-r^n} - \frac{r^n}{1-r^n}\right) \\ &\ge 2S(r - 2r^n) \\ &\ge 2S\left(\frac{1}{2} + \epsilon - \frac{\epsilon}{2}\right) \\ &= S(1+\epsilon) . \end{aligned}$$

(The inequalities follow from the fact that $1 - r^n \le 1$ and since $r^n \le 1/2$ and $4r^n < \epsilon$ by the choice of n.)

We now proceed to prove that $C(S', F')$ holds at time $2S + n$. Let us call the packets described in part (1) of the definition of the adversary *old* packets, the packets described in part (2) *new short* packets, and the packets described in parts (3) and (4) *new long* packets.

We start with the following straightforward property.

CLAIM 3.8. *In each step in the time interval* $[1, 2S]$*, one old packet crosses* a'*.*

Proof. Since there are S old packets in the buffers of edges e_i, there are no other packets in these buffers, and none of them is empty at time 0, then these S packets will arrive at the tail of a' one in each time step in time interval $[1, S]$. The S packets stored at the tail of a at time 0 will arrive at the tail of a', one in each time step, in the time interval $[n, S + n]$. Since $S > n$, the claim follows. □

The next claim shows that old packets cross edges e'_i at rates that decrease as i grows. This is due to the injection of the new short packets.

CLAIM 3.9. *The following holds for any edge* e'_i*,* $i \in [1, n]$*.*

(1) *At times* $[0, i]$*, no packet arrives at the tail of* e'_i*.*
(2) *At times* $[i+1, 2S+i]$*, old packets arrive at the tail of* e'_i *at rate* R_i*.*
(3) *At time* $i+2S+1$*, there are no new short packets in the buffer of* e'_i*.*

Proof. Part (1) is straightforward by the fact that old packets must cross at least $i+1$ edges before they arrive at the tail of e'_i and because no new packet is injected for e'_i before time i. Part (2) is proven by induction on i. For the basis $i = 1$ we have that packets arrive at the tail of e'_1 at rate $R_1 = 1$ by Claim 3.8. For the induction step, let $i > 1$. The induction hypothesis says that packets arrive at the tail of e'_{i-1} at rate R_{i-1}. By part (2) of the definition of the adversary, new packets are injected at the tail of e'_{i-1} at rate r. Note that $R_{i-1} + r > 1$. Since the queue policy is FIFO, it follows that old packets cross e'_{i-1}, and hence arrive at the tail of e'_i, at rate $\frac{R_{i-1}}{R_{i-1}+r}$. By (3.1) this is exactly R_i. This proves part (2). To see that part (3) is true, note that as a consequence of part (2), we have that short new packets cross e'_i at rate $\frac{r}{R_i+r}$. The last short new packet for e'_i is injected at time $i + t_i = i + \frac{2S}{r+R_i}$, in which time there are $t_i(r + R_i - 1)$ packets in the buffer of e'_i. Using the definition of t_i, it follows that all new short packets of e'_i will be absorbed by time

$$i + t_i + t_i(r + R_i - 1) = i + t_i(r + R_i) = i + 2S \ . \qquad \square$$

Using the above claims, we show that $C(S', F')$ holds at time $2S + n$. We start with part (1) of $C(S', F')$.

Claim 3.10. *At time* $2S+n$*, there is a total of* S' *old packets stored in the buffers of edges* e'_i*.*

Proof. By Claim 3.9, $2S \cdot R_n$ old packets cross a'' by time $2S + n$. On the other hand, by Claim 3.8, all the $2S$ old packets crossed a' by time $2S$. The claim follows. $\square$

Next, we prove that part (2) of $C(S', F')$ holds.

Claim 3.11. *If* $S > S_0$*, then none of the buffers of* e'_i *is empty at time* $2S + n$*. Moreover, the route of any packet stored in* e'_i *at that time is* $e'_i, \ldots, e'_n, a''$*.*

Proof. The claim on the remaining routes is obvious from the construction. We now prove that the buffer of e'_i is not empty. The last short packet for e'_i is injected in e'_i at time $i + t_i$, and, as argued in the proof of Claim 3.9, it crosses e'_i at time $2S + i$. Hence all packets that arrive at the buffer of e'_i in the time interval $[i + t_i, 2S + i]$ are still in the buffer of e'_i at time $2S + i$. All these packets are old packets that arrive from e_{i-1}. By Claim 3.9, there are $(2S - t_i)R_i$ such packets. Let $Q_i \stackrel{\text{def}}{=} (2S - t_i)R_i$ be the number of packets in the buffer of e'_i at time $2S + i$. Note that by definition, $t_i \le t_{i+1}$ and $R_i \ge R_{i+1}$, and hence $Q_i \ge Q_{i+1}$ for $1 \le i < n$. In addition, since only $n - i$ packets may leave the buffer of e'_i in the time interval $[2S + i, 2S + n]$, it is sufficient to prove that $Q_n \ge n$. Substituting the values we get

$$\begin{aligned} Q_n &= (2S - t_n)R_n \\ &= 2SR_n - t_nR_n \\ &= 2S\left(R_n - \frac{R_n}{r + R_n}\right) \\ &= 2S\left(R_n - R_{n+1}\right) \ . \end{aligned}$$

Since $S \ge S_0 > \frac{n}{2(R_n - R_{n+1})}$, we get that $Q_n \ge n$. $\square$

We now prove that part (3) of $C(S', F')$ holds.

Claim 3.12. *The number of packets at the tail of* a' *at time* $2S + n$ *is* S'*.*

Proof. First, observe that in time interval $[1, S+n]$ the number of packets that arrive at the tail of a' is exactly $2S$, and they start arriving at time 1. Therefore at time $S+n$ there are exactly $S-n$ packets in the buffer of a'. In addition, by part (3) of the definition of the adversary, rS new long packets are injected at the tail of a in the time interval $[1, S]$. These packets start crossing a at time $S+1$, since they are queued behind the S old packets stored in a at time 0. Hence the new long packets start arriving at a' at time $S+n+1$. In addition, part (4) of the definition of the adversary says that X new long packets are injected at the tail of a' during time interval $[S+n, 2S+n]$. In conclusion, there are $X+rS$ new long packets arriving at a' in the interval $[S+n, 2S+n]$. Together with the $S-n$ packets stored at the tail of a' at time $S+n$, we have that at time $2S+n$, the number of packets stored in the buffer of a' is exactly $rS+X-n=S'$, by definition of X. All these packets have the paths that are required by $C(S', F')$. □

To conclude the proof of Lemma 3.6, we argue that F is empty at time $2S+n$. This follows from the fact that there are no injections into edges of F during time interval $[0, 2S+n]$ and that all the $2S$ packets present in F at time 0 arrive at the tail of the ingress of F' by time $S+n$. □

3.3. Putting the gadgets together. In this section we describe how to construct the overall adversary, using the gadget adversary described in section 3.2, and a few other simple adversaries used to glue things together.

The idea of the proof is to use a sufficiently long daisy chain of gadgets that blows up the queue size by a sufficiently large factor (that depends on the length of the chain and r) and then "stitch together" the egress of the chain to its ingress, getting a queue of fresh packets. The stitching process loses a fraction (that depends on r) of the queue size, but this loss is more than compensated by the chain of gadgets.

Fix $r = \frac{1}{2} + \epsilon$ for $\epsilon > 0$ and S_0 and n as in the proof of Lemma 3.6. Consider the graph F_n^M that consists of a daisy chain of M F_n gadgets, where M is a parameter. Let the kth gadget be denoted by $F(k)$ for $1 \le k \le M$. We now prove the following lemma.

LEMMA 3.13. *Let M be a positive integer, and consider the graph F_n^M. If for some time τ we have that all packets present in the network were injected after time τ_0, and*

- *$C(S, F(1))$ holds at time τ for $S \ge S_0$,*
- *there are no other packets in F_n^M at time τ, and*
- *the edges of $F_n^M \setminus F(1)$ were not used by any injection in the time interval $[\tau_0 - \lceil 1/r \rceil, \tau]$,*

then there is a rate-r adversary such that at some time $t > \tau$, there are S' packets at the egress of F_n^M, for $S' \ge S(1+\epsilon)^{M-1}/2$, and there are no other packets in F_n^M.

Proof. We first prove the following claim.

CLAIM 3.14. *Let $1 \le i \le n$. If at time τ we have that all packets present in the network were injected after time τ_0, and*

- *$C(S, F(1))$ holds for $S \ge S_0$,*
- *there are no other packets in $F(1), \ldots, F(i)$, and*
- *the edges of $F(2), \ldots, F(M)$ were not used by any injection in the time interval $[\tau_0 - \lceil 1/r \rceil, \tau]$,*

then there is a rate-r adversary and time $t_i \ge \tau$ such that

- *$C(S', F(i))$ holds for $S' \ge S(1+\epsilon)^{i-1}$ at time t_i,*
- *there are no other packets in F_n^M, and*

- *the edges of* $F(i+1), \ldots, F(M)$ *were not used by any injection in the time interval* $[\tau_0 - \lceil 1/r \rceil, t_i]$.

Proof. The proof is by induction on i. For $i = 1$ the claim is trivial with $t_1 = \tau$. For the induction step, assume that the lemma holds for $1 < i < M$, i.e., that there exists an adversary $\mathcal{A}_i$ and time t_i such that at time t_i, $C(S_i, F(M))$ holds for $S_i \geq S(1+\epsilon)^{i-1}$. Consider now the subgraph that consists of $F(i)$ and $F(i+1)$. By the induction hypothesis, we may apply Lemma 3.6 to know that there exists an adversary $\mathcal{A}$ such that at time $t_i + 2S_i + n$, $C(S', F(i+1))$ holds for $S' \geq S_i(1+\epsilon) \geq S(1+\epsilon)^i$. We note that the packets injected by $\mathcal{A}$ (as specified in Lemma 3.6) do not use any edge in $F(i+2), \ldots, F(M)$ and that the application of this adversary leaves $F(i)$ empty of packets. This proves the claim with $t_{i+1} = t_i + 2S_i + n$ and the adversary that results from concatenating the adversaries $\mathcal{A}_i$ and $\mathcal{A}$. □

To complete the proof of Lemma 3.13, we observe that if at time t we have that $C(S, F_n)$ holds for some gadget F_n and $S \geq S_0$, and if no injections are done in the interval $[t, t+S+n]$, then at time $t+S+n$ there are at least $S/2$ packets queued at the egress of F_n. This is true since during time interval $[t+1, t+S+n]$ exactly $2S$ packets arrive at the tail of the egress of F_n, and, therefore, at time $t+S+n$ there are $S - n \geq S_0 - n \geq S/2$ packets in the egress buffer.

Note that a packet may be rerouted in the whole construction at most $M-1$ times: once for each gadget $F(2), \ldots, F(M)$. This completes the proof of Lemma 3.13. □

We now specify two more constructions. The first shows how to establish $C(S, F_n)$ starting from a state in which the only packets in F_n are in the buffer of the ingress of F_n. The second construction shows how to replace a queue of packets with another queue of *fresh* packets. This is necessary so we can stitch the end of the daisy chain to its beginning.

We now claim the existence of an adversary that establishes $C(S, F_n)$ starting from a single buffer. The construction is a variant of the adversary presented in the proof of Lemma 3.6.

LEMMA 3.15. *For any* $\epsilon > 0$, *let* n *and* S_0 *be as in Lemma* 3.6. *Let* $S > S_0$, *and let* τ *be a time step. Suppose that at time* τ *all the packets in the network are* $2S$ *packets stored in the ingress edge of* F_n, *they all have remaining routes of length* 1, *and they were all injected after time* τ_0 *for some* τ_0. *If the other edges of* F_n *were not used by any injection in the time interval* $[\tau_0 - \lceil 1/r \rceil, \tau]$, *then there is a rate-*r *adversary for* $r = \frac{1}{2} + \epsilon$ *such that at time* $\tau + 2S + n$ *condition* $C(S', F_n)$ *holds for* $S' \geq S(1+\epsilon)$.

Proof. Let us again assume for convenience of notation that $\tau = 0$. We use the notations and definitions of t_i and R_i from the proof of Lemma 3.6. We also define $S' = 2S(1 - R_n)$. The adversary is defined as follows.

(1) Extend the route of the packets stored in the ingress edge a to be $a, e_1, e_2, \ldots, e_n, a'$.
(2) For each $1 \leq i \leq n$, inject packets at rate r with the single edge route e_i in the time interval $[i, t_i]$.
(3) In the first $(S'+n)/r$ time steps of time interval $[1, 2S]$ inject $S'+n$ packets at rate r. The first n packets have path of length 1 (i.e., a only), and the rest have the path $a, f_1, \ldots, f_n, a'$. Observe that indeed $(S'+n)/r \leq 2S$, by the choice of n and S_0.

We first note that this is a rate-r adversary by Lemma 3.3. We now prove that $C(S', F_n)$ holds at time $2S+n$. First, observe that in each step of the interval $[1, 2S]$,

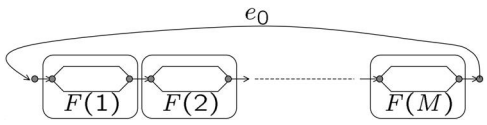


FIG. 3.2. *The graph used in the proof of Theorem* 3.17. *The edge between $F(i)$ and $F(i+1)$ is the egress of $F(i)$ and the ingress of $F(i+1)$, and it is part of both $F(i)$ and $F(i+1)$.*

a single packet crosses a. By the same arguments as those in the proofs of Claims 3.9, 3.10, and 3.11 (applied here to F_n, instead of F' there), we have that at time $2S+n$ there are S' packets in the buffers of edges $e_1, \ldots, e_n$, that none of these buffers is empty, and that the packets in e_i have remaining routes $e_i, e_{i+1}, \ldots, e_n, a'$. Next, consider a. After $2S$ time steps, all old packets leave a; after additional n time steps all packets with path of length 1 injected in step (3) disappear too, and therefore, at time $2S+n$, we have exactly S' packets in a, with remaining routes $a, f_1, \ldots, f_n, a'$, as required. □

We now show the existence of an adversary that replaces a queue of old packets with another (smaller) queue of fresh packets. To do this, we consider a graph of three edges in series, called a_0, a_1, and a_2. The routes that will be traversed by old packets will all end at a_0, and the fresh packets all start at the tail of a_2. (We use three edges instead of two so as to avoid cyclic routes in our final construction.)

LEMMA 3.16. *Suppose that at time τ there are S packets stored in the buffer of a_0 with remaining routes of length* 1. *Then for any $r > 0$ there exists a rate-r adversary such that at time $\tau + S + rS + r^2S$ there are r^3S packets stored in the buffer of a_2 and there are no other packets in the network. Moreover, all the packets stored in the buffer of a_2 were injected at the tail of a_2 after time $\tau + S$.*

Proof. Let us assume for convenience of notation that $\tau = 0$. Call the packets that exist in the network at time 0 *old packets*. The execution is as follows.

(1) In the time interval $[1, S]$, rS packets are injected at the tail of a_0. These packets have routes a_0, a_1, a_2. All these packets are queued behind the old packets, and they start to move only at time S.
(2) In the time interval $[S+1, S+rS]$, r^2S packets are injected at the tail of a_2. These packets mix with the packets that were injected in step (1). At time $S + rS$, there is a queue of r^2S packets waiting for a_2, and no other packets exist in the network.
(3) In the time interval $[S + rS, S + rS + r^2S]$, r^3S new packets are injected at the tail of a_2. These packets are queued behind the packets injected in steps (1) and (2).

Note that by time $S+rS+r^2S$, all packets from steps (1) and (2) are absorbed. □

We are now ready to prove our main result. Note that the assumption of a specific initial state does not restrict the generality of the statement (see, e.g., [4]).

THEOREM 3.17. *For every $\epsilon > 0$ there exists a graph G_ϵ, a rate-r adversary for $r = \frac{1}{2} + \epsilon$, and an initial configuration such that FIFO is unstable on G_ϵ under that adversary starting from that initial configuration.*

Proof. The graph is defined as follows. Let S_0 and n be as required by Lemma 3.6 for ϵ. Choose M such that $\frac{r^3(1+\epsilon)^M}{4} > 1$. The graph consists of F_n^M (i.e., M daisy-chained gadgets), with one additional edge called e_0 connecting the head of the egress edge of the last gadget in the chain ($F(M)$) to the ingress edge of the first gadget in the chain ($F(1)$). See Figure 3.2.

In the initial configuration, there are $S^* > 2S_0$ packets in the ingress edge of

$F(1)$, all with paths of length 1 (i.e., their paths are composed of the ingress of $F(1)$). The adversary is defined by an iterative construction that works as follows. Let $S_1 = S^*$.

(1) Apply the adversary of Lemma 3.15 to get a configuration where $C(S_2, F(1))$ holds for $S_2 \geq \frac{S_1}{2}(1+\epsilon)$.
(2) Apply the adversary of Lemma 3.13 to get a configuration where S_3 packets are stored in the egress of $F(M)$ for $S_3 \geq S_2 \frac{(1+\epsilon)^{M-1}}{2}$.
(3) Apply the adversary of Lemma 3.16 to the three-edge path that consists of the egress of $F(M)$, then e_0, and then the ingress of $F(1)$. This results in S_4 packets stored at the tail of the ingress of $F(1)$, all with paths of length 1, for $S_4 \geq r^3 S_3$. Let $S_1 \leftarrow S_4$, and go to step (1).

We first claim that the above construction is indeed a valid rate-r adversary. We claim inductively that the conditions that allow the construction of each adversary by Lemmas 3.15, 3.13, and 3.16 hold when these constructions are applied. We base the proof for each iteration of the adversary on the following condition, which we will prove by induction on time to hold at the start of each iteration. The condition is that when the iteration starts at time τ, then it holds that all the buffers in the network are empty except the buffer of the ingress of $F(1)$; that all packets in this buffer have routes of length 1; and that in time interval $[\tau_0 - \lceil 1/r \rceil, \tau]$ there were no injections of packets requiring any other edge in the network, where τ_0 is the earliest injection time of any packet residing in the buffer of the ingress of $F(1)$ at time τ. This condition clearly holds for time $\tau = 0$ when we start with the initial configuration as described above.

Now assume that the above condition holds at time τ when an iteration starts. Then the conditions of Lemma 3.15 hold. We therefore can apply the adversary of Lemma 3.15. The resulting situation is the situation as required by the conditions of Lemma 3.13. Observe that the adversary of Lemma 3.15 does not use any edge in the network beyond the edges of $F(1)$; therefore the conditions of that lemma on the nonuse of the edges of $F_n^M \setminus F(1)$ hold. We can therefore apply the adversary of Lemma 3.13. The resulting situation is the situation in the conditions of Lemma 3.16. Observe also that by Lemma 3.13 there are no other packets in the network at that time. We can now apply the adversary of Lemma 3.16. This results in a set of packets in the buffer of the ingress of $F(1)$. Note that now there are no other packets in the network. Further observe that all the packets at the ingress of $F(1)$ were injected at least S time steps into the activation of the adversary of Lemma 3.16 and that once they start being injected, no other packet is injected. Therefore it follows that no edge of the graph, except the ingress of $F(1)$, was used by the adversary after time $\tau_0 - \lceil 1/r \rceil$, where τ_0 is the earliest injection time of the packets at the ingress of $F(1)$. We therefore have that the condition for the start of the iteration holds again.

We note that no packet is rerouted more than M times: once in step (1) and at most $M-1$ times in step (2).

Finally, we show that S_1 grows unboundedly under this adversary. After step (1) is executed, we have that $S_2 \geq \frac{S_1}{2} \cdot (1+\epsilon)$. Hence, after step (2) we have that $S_3 \geq S_2 \frac{(1+\epsilon)^{M-1}}{2} \geq S_1 \frac{(1+\epsilon)^M}{4}$. Finally, after step (3), we have that the number of packets stored at the tail of the ingress edge of $F(1)$ is $S_4 \geq r^3 S_3 \geq S_1 \frac{r^3(1+\epsilon)^M}{4}$. By the choice of M we have that $S_4 > S_1$. □

4. Stability under low injection rates. In this section we show that FIFO, and in fact any greedy protocol, is stable if the injection rate is below some threshold.

We start with the case where the network is initiated with empty buffers. We later consider the case where the adversary starts the system with an arbitrary initial configuration of an arbitrary set of packets in the buffers.

We start with the case where the network starts with empty buffers. We prove that any network is stable with *any greedy* protocol in the face of a (w, r) adversary, if $r \leq 1/(d+1)$, where d denotes the length (in edges) of the longest path followed by any packet. In particular, we prove below that any packet stays in any one queue no more that $\lfloor wr \rfloor$ time steps. For a certain class of protocols, which includes the protocol FIFO, the bound can be improved to $1/d$.

THEOREM 4.1. *For any network, if the sequence of packets is injected by a (w, r) adversary, with $r \leq 1/(d+1)$, and the schedule is a greedy schedule, then no packet stays in the same buffer more than $\lfloor wr \rfloor$ time steps.*

Proof. We prove, by induction on t, that any packet that arrives at a buffer at time step t leaves this buffer by time $t + \lfloor wr \rfloor$.

The base of the induction is any $t \leq dwr+1$. Let p be a packet that arrives at the buffer at the tail of edge e at time $t \leq dwr+1$. Assume towards a contradiction that p is in the same buffer at the end of time step $t+\lfloor wr \rfloor$. This means that for each of the $\lfloor wr \rfloor$ time steps in $[t+1, t+\lfloor wr \rfloor]$ some other packet was sent over edge e (since we consider a *greedy* protocol). Therefore we can identify $\lfloor wr \rfloor + 1$ packets that require edge e and are injected into the network by the end of time step $t + \lfloor wr \rfloor - 1$ (these are the packet p itself and the $\lfloor wr \rfloor$ packets that were sent over e). Since $t \leq dwr+1$, we have $t + \lfloor wr \rfloor - 1 \leq (d+1)wr$. By the definition of the adversary the number of packets that require e and are injected by the end of any time step $t' \leq (d+1)wr$ is at most $\lceil (d+1)r \rceil \lfloor wr \rfloor$. Since we assume $r \leq 1/(d+1)$ this is at most $\lfloor wr \rfloor$. This is a contradiction to the fact that we identified $\lfloor wr \rfloor + 1$ packets.

We now prove the claim for any $t > dwr+1$. This is done based on the induction hypothesis that for any packet that arrives at some buffer at time $t' < t$, this packet leaves the buffer by time step $t' + \lfloor wr \rfloor$. Let p be a packet that arrives at the buffer at the tail of edge e at some time step t. Consider any packet that requires edge e and was injected by time step $t - d\lfloor wr \rfloor$. Using the induction hypothesis we know that such a packet left the buffer into which it was injected by time step $t - d\lfloor wr \rfloor + \lfloor wr \rfloor$, left the next buffer by time step $t - d\lfloor wr \rfloor + 2\lfloor wr \rfloor$, and left the ith buffer on its path by time step $t - d\lfloor wr \rfloor + i\lfloor wr \rfloor$. It therefore arrived at its destination by time step $t - d\lfloor wr \rfloor + d\lfloor wr \rfloor = t$ (since the length of its path is at most d, and all its "arrival times" are earlier than t, so the induction hypothesis holds). It follows that any packet that can delay packet p from going over edge e must be injected at time step $t - d\lfloor wr \rfloor + 1$ or later. Now assume towards a contradiction that packet p is still at the tail of edge e at the end of time step $t + \lfloor wr \rfloor$. That is, there are $\lfloor wr \rfloor$ other packets that crossed edge e in $[t+1, t+\lfloor wr \rfloor]$. As before, this identifies $\lfloor wr \rfloor + 1$ distinct packets that require edge e, are present in the network at the end of time step t or later, and are injected by time step $t + \lfloor wr \rfloor - 1$. However, we know that any packet injected by time step $t - d\lfloor wr \rfloor$ already left the network by the end of time step t. Therefore those $\lfloor wr \rfloor + 1$ packets must have been injected in $[t - d\lfloor wr \rfloor + 1, t + \lfloor wr \rfloor - 1]$. There are $\lfloor wr \rfloor (d+1) - 1$ time steps in this interval; therefore the number of packets that require e and can be injected during this interval is bounded by $\lceil (d+1)r \rceil \lfloor wr \rfloor$. Since $r \leq 1/(d+1)$ this is at most $\lfloor wr \rfloor$, a contradiction. ∎

For protocols where a packet arriving at a certain buffer at time t has priority over any packet injected after time t, we can relax the condition that $r \leq 1/(d+1)$ to be $r \leq 1/d$. Note that among such protocols are the protocols FIFO and LIS.

Specifically, we define the following concept.

DEFINITION 4.2. *A* time priority *protocol is a greedy protocol under which a packet arriving at a buffer at time t has priority over any other packet that is injected after time t.*

For time priority protocols, we have the following.

THEOREM 4.3. *For any network, if the sequence of packets is injected by a (w, r) adversary, with $r \leq 1/d$, and the protocol is a time priority protocol, then no packet stays in the same buffer more than $\lfloor wr \rfloor$ time steps.*

The proof of Theorem 4.3 is the same as the proof of Theorem 4.1 with one change applied at two places: in the present case, when assuming towards a contradiction that packet p is still in the same buffer at the end of time step $t+\lfloor wr \rfloor$ and identifying the packets that cause this delay, we know that those packets must have been injected no later than time step t (rather than time $t + \lfloor wr \rfloor - 1$). This is because packets injected after time step t will not delay packet p if the protocol is a time priority protocol. This allows us to prove the lemma with the relaxed condition that $r \leq 1/d$. For completeness we give below the full proof.

Proof. We prove that any packet that arrives at any buffer at time step t leaves this buffer by time step $t + \lfloor wr \rfloor$. The proof is by induction on t.

We prove the base of the induction for any $t \leq dwr$. Let p be a packet that arrives at the buffer at the tail of edge e at time $t \leq dwr$. Assume towards a contradiction that p is in the same buffer at the end of time step $t + \lfloor wr \rfloor$. This means that during the $\lfloor wr \rfloor$ time steps in $[t+1, t+\lfloor wr \rfloor]$ some other packet was sent over edge e (since we consider a *greedy* protocol). We can therefore identify $\lfloor wr \rfloor + 1$ packets that require edge e (these packets are the packet p itself and the $\lfloor wr \rfloor$ packets that were sent over e). These packets must have been injected into the system by the end of time step t; any packet injected after t will not delay p according to a time priority protocol. Now, by the definition of the adversary the number of packets that require e and are injected by the end of any time step $t \leq dwr$ is at most $\lceil dr \rceil \lfloor wr \rfloor$. Since we assume $r \leq 1/d$ this is at most $\lfloor wr \rfloor$. This is a contradiction to the fact that we identified $\lfloor wr \rfloor + 1$ packets.

We now prove the claim for any $t > dwr$. This is done based on the induction hypothesis that for any packet that arrives at some buffer at time $t' < t$, this packet leaves this buffer by time step $t' + \lfloor wr \rfloor$. Let p be a packet that arrives at the buffer at the tail of edge e at some time step t. Consider any packet that requires edge e and was injected by time step $t - d\lfloor wr \rfloor$. Using the induction hypothesis we know that such a packet left the buffer into which it was injected by time step $t - d\lfloor wr \rfloor + \lfloor wr \rfloor$, left the next buffer by time step $t - d\lfloor wr \rfloor + 2\lfloor wr \rfloor$, and left the ith buffer on its path by time step $t - d\lfloor wr \rfloor + i\lfloor wr \rfloor$. It therefore arrived at its destination by time step $t - d\lfloor wr \rfloor + d\lfloor wr \rfloor = t$ (since the length of its path is at most d, and all its "arrival times" are earlier than t, so the induction hypothesis holds). It follows that any packet that can delay packet p from going over edge e must be injected at time step $t - d\lfloor wr \rfloor + 1$ or later. Now assume towards a contradiction that packet p is still at the tail of edge e at the end of time step $t + \lfloor wr \rfloor$. That is, there are $\lfloor wr \rfloor$ other packets that crossed edge e in $[t + 1, t + \lfloor wr \rfloor]$. As before, this identifies $\lfloor wr \rfloor + 1$ distinct packets that require edge e and are injected by the end of time step t; any packet injected after t will not delay p since the protocol is a time priority protocol. However, we know that any packet injected by time step $t - d\lfloor wr \rfloor$ already left the network by the end of time step t. Therefore those $\lfloor wr \rfloor + 1$ packets must have been injected in $[t - d\lfloor wr \rfloor + 1, t]$. There are $\lfloor wr \rfloor d$ time steps in this interval; therefore the

number of packets that require e and can be injected during this interval is bounded by $\lceil dr \rceil \lfloor wr \rfloor$. Since $r \leq 1/d$ this is at most $\lfloor wr \rfloor$, a contradiction. □

We now show that similar results hold when the adversary is allowed to initiate the system with an arbitrary set of packets in the network. In this case, the requirement for the rate is that it is less than (rather than at most) $1/(d+1)$ (or $1/d$ for time priority protocols). This follows from the fact that if an adversary starts the system with some initial set of packets and then injects packets as a (w, r) adversary, then the same sequence of packets can be given by an adversary that starts the system with an empty initial configuration and then injects packets as a (w^*, r^*) adversary for any $r^* > r$ and an appropriately chosen w^*. In the following we call an initial configuration an *S-initial-configuration*, for $S \geq 0$, if S is the maximum, over the edges $e \in E$, of the number of packets requiring e, in the initial configuration. We now give the following observation.

OBSERVATION 4.4. *Any sequence of packets given by a (w, r) adversary that starts with an S-initial-configuration can be given by a (w^*, r^*) adversary that starts with a 0-initial-configuration (i.e., with empty buffers) for any $r^* > r$ and $w^* = \lceil \frac{S+w+1}{r^*-r} \rceil$.*

Proof. We show that indeed a (w^*, r^*) adversary can inject the same sequence of packets without the need of an initial nonempty configuration. The new adversary will start with an empty configuration, will inject the packets of the initial state in time step 1, and will later inject in every time step t the same packets that the old adversary injected in time step $t-1$. By construction the new adversary starts with an empty configuration. It remains therefore to show that it is a valid (w^*, r^*) adversary.

To see that we show that in every consecutive w^* time steps the adversary injects at most $\lfloor w^* r^* \rfloor$ packets requiring any particular edge. We first note that

$$\lfloor w^* r^* \rfloor - w^* r > w^* r^* - 1 - w^* r = w^*(r - r^*) - 1 = \left\lceil \frac{S + w^* + 1}{r^* - r} \right\rceil (r - r^*) - 1 \geq S + w \, .$$

Therefore,

$$\lfloor w^* r^* \rfloor > S + w + w^* r \geq S + wr + w^* r \geq S + \left\lceil \frac{w^*}{w} \right\rceil \lfloor wr \rfloor \ .$$

We have two types of time intervals: one time interval that includes time step 1 (i.e., $[1, w^*]$) and any other time interval of w^* consecutive time steps. For the latter case, the new adversary injects in some time interval $[\tau, \tau + w^* - 1]$ the same packets as the old adversary injected in time interval $[\tau - 1, \tau - 1 + w^* - 1]$. For every edge e the maximum number of packets requiring e injected by the (w, r) adversary is at most $\lceil \frac{w^*}{w} \rceil \lfloor wr \rfloor$, so the new adversary does not violate its constraints. For time interval $[1, w^*]$, the number of packets requiring any particular edge given by the (w, r) adversary, either in the initial configuration or the first w^* time steps, is at most $S + \lceil \frac{w^*}{w} \rceil \lfloor wr \rfloor$. □

The following two corollaries now follow immediately from the above observation and from Theorems 4.1 and 4.3.

COROLLARY 4.5. *For any network, if the system is started with an S-initial-configuration, the sequence of packets is injected by a (w, r) adversary with $r < 1/(d+1)$ and the schedule is a greedy schedule, then no packet stays in the same buffer more than $\lfloor \lceil \frac{S+w+1}{\frac{1}{d+1} - r} \rceil \cdot \frac{1}{d+1} \rfloor$ time steps.*

COROLLARY 4.6. *For any network, if the system is started with an S-initial-configuration, the sequence of packets is injected by a (w, r) adversary with $r < 1/d$*

and the protocol is a time priority protocol, then no packet stays in the same buffer more than $\lfloor \lceil \frac{S+w+1}{\frac{1}{d}-r} \rceil \cdot \frac{1}{d} \rfloor$ *time steps.*

5. Conclusions. In this paper we show upper and lower bounds on the rates at which FIFO is stable. These results improve upon previous bounds [4, 11, 15]. We note that our lower bounds use shortest-paths (and hence noncircular) routes.

We also show that any greedy protocol is always stable against a (w, r) adversary for $r < 1/(d+1)$, where d is the length of the longest route used (or $r < 1/d$ for a certain class of protocols). Results in [7] show that the protocol FTG (and in fact also LIFO and NTS) can be unstable for arbitrarily low rates. The proofs there use a network and a set of paths such that in order to show that FTG is unstable for rate r, packets with paths of length $16/r$ are used. In view of these results, our bounds on r, in terms of d, are optimal up to a small constant factor. Furthermore, our stability results indicate that in order to show that FIFO can be unstable at arbitrarily low rates, one would need correspondingly long paths for the packets, as opposed to the (small) constant size networks (and hence constant size packet routes) used to prove previous results on the instability of FIFO.

The technique we use for the instability result, of constructing gadgets and chaining them, can be applied to various gadgets. For example, one can extract a gadget structure from the constructions of [4] or [11], compose them as in Theorem 3.17, and improve on the original bounds. Conceptually, our lower bound consists of two elements: the chain idea and a "good" gadget. We believe that this technique may lead to further improvements.

Appendix: Asymptotic bounds for Lemma 3.6. In this appendix we give asymptotic bounds for the parameters n and S_0 used in Lemma 3.6. Specifically, we show the following. Let $\epsilon > 0$ be given. We define the following quantities:

$$r = \frac{1}{2} + \epsilon \ , \tag{5.1}$$

$$R_i = \frac{1-r}{1-r^i} \quad \text{for all } i \geq 0 \ , \tag{5.2}$$

$$n = \max\left(\frac{\log\epsilon - 2}{\log r}, \quad 1 - \frac{1}{\log r}\right) \ , \tag{5.3}$$

$$S_0 = \max\left(2n, \quad \frac{n}{2(R_n - R_{n+1})}\right) \ . \tag{5.4}$$

We prove that $n = \Theta(\log \frac{1}{\epsilon})$ and $S_0 = \Theta(nr^{-n}) = \Theta(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ when $\epsilon \to 0^+$ (i.e., since all quantities are functions of ϵ, we may consider the case when $\epsilon \to 0^+$).

We remark that in what follows we do not attempt to get tight constant factors. We use only crude estimates that, however, allow us to prove tight asymptotic bounds.

We start with n. Note first that by (5.1), $\log\epsilon < \log r$, and hence $\log\epsilon - 2 < \log r - 1$. For $\epsilon < \frac{1}{2}$, we also have $\log r < 0$, and hence $\frac{\log\epsilon-2}{\log r} > 1 - \frac{1}{\log r}$. It follows from (5.3) that for $\epsilon < 1/2$, $n = \frac{\log\epsilon-2}{\log r}$. Moreover, for $0 < \epsilon < 1/\sqrt{2} - 1/2$, we have $1/2 < r < 1/\sqrt{2}$, and therefore

$$\frac{\log\epsilon - 2}{-1} \ < \ n \ < \ \frac{\log\epsilon - 2}{-1/2} \ .$$

The latter bounds are equivalent to

$$\log\frac{1}{\epsilon} + 2 < \ n \ < 2\log\frac{1}{\epsilon} + 4 \ , \tag{5.5}$$

and hence, for $\epsilon \to 0^+$, we have that

$$n = \Theta\left(\log \frac{1}{\epsilon}\right) . \tag{5.6}$$

We now consider S_0. To show the claim, we bound S_0 for $0 < \epsilon < 1/4$. We start by estimating the difference $R_n - R_{n+1}$. Using (5.2) we have

$$\begin{aligned} R_n - R_{n+1} &= \frac{1-r}{1-r^n} - \frac{1-r}{1-r^{n+1}} \\ &= \frac{(1-r)(1-r^{n+1}) - (1-r)(1-r^n)}{(1-r^n)(1-r^{n+1})} \\ &= \frac{1 - r^{n+1} - r + r^{n+2} - 1 + r^n + r - r^{n+1}}{(1-r^n)(1-r^{n+1})} \\ &= r^n \cdot \frac{1 + r^2 - 2r^{n+1}}{(1-r^n)(1-r^{n+1})} . \end{aligned} \tag{5.7}$$

Using (5.7), we bound $(R_n - R_{n+1})/r^n$ from both sides by constants as follows. For $0 < \epsilon < 1/4$ we have $\frac{1}{2} < r < \frac{3}{4}$, and by (5.5) we have $n > 4$. Hence

$$1 < 1 + r^2 - 2r^{n+1} \ < \ \frac{1 + r^2 - 2r^{n+1}}{(1-r^n)(1-r^{n+1})} \ < \ \frac{1 + r^2 - 2r^{n+1}}{\frac{1}{4}} < 8 . \tag{5.8}$$

Therefore, $R_n - R_{n+1} = \Theta(r^n)$ for $\epsilon \to 0^+$. Moreover, for $0 < \epsilon < 1/32$ we have by (5.7), (5.8), and (5.5) that $R_n - R_{n+1} < \frac{1}{4}$, and hence $2n < \frac{n}{2(R_n - R_{n+1})}$. It therefore follows from (5.4) that for $\epsilon \to 0^+$,

$$S_0 \ = \ \frac{n}{2(R_n - R_{n+1})} \ = \ \Theta(nr^{-n}) , \tag{5.9}$$

as desired. Finally, note that by (5.3) we have that for $\epsilon < 1/2$,

$$nr^{-n} = nr^{-\frac{\log \epsilon - 2}{\log r}} = 4n2^{-\log \epsilon} = \frac{4n}{\epsilon} . \tag{5.10}$$

Combining (5.6), (5.9), and (5.10), we conclude that $S_0 = \Theta\left(\frac{1}{\epsilon} \log \frac{1}{\epsilon}\right)$ for $\epsilon \to 0^+$.

REFERENCES

[1] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén, *Adaptive packet routing for bursty adversarial traffic*, J. Comput. System Sci., 60 (2000), pp. 482–509.

[2] C. Àlvarez, M. Blesa, and M. Serna, *Universal stability of undirected graphs in the adversarial queueing model*, in Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures, Winnepeg, MB, Canada, 2002, pp. 183–197.

[3] M. Andrews, *Instability of FIFO in session oriented networks*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, pp. 440–447.

[4] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu, *Universal stability results for greedy contention-resolution protocols*, J. ACM, 48 (2001), pp. 39–69.

[5] M. Andrews and L. Zhang, *The effects of temporary sessions on network performance*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, pp. 448–457.

[6] A. Borodin, *private communication*, 2001.

[7] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. Williamson, *Adversarial queuing theory*, J. ACM, 48 (2001), pp. 13–38.

[8] R. Bhattacharjee and A. Goel, *Instability of FIFO at arbitrarily low rates in the adversarial queueing model*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, Cambridge, MA, 2003, pp. 160–167.

[9] R. Cruz, *A calculus for network delay, Part* I*: Network elements in isolation*, IEEE Trans. Inform. Theory, 37 (1991), pp. 114–131.

[10] R. Cruz, *A calculus for network delay, Part* II*: Network analysis*, IEEE Trans. Inform. Theory, 37 (1991), pp. 132–141.

[11] J. Díaz, D. Koukopoulos, S. Nikoletseas, M. Serna, P. Spirakis, and D. Thilikos, *Stability and non-stability of the FIFO protocol*, in Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architectures, Crete, Greece, 2001, pp. 48–52.

[12] D. Gamarnik, *Stability of adversarial queues via fluid model*, in Proceedings of the 39th Annual Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, pp. 60–70.

[13] D. Gamarnik, *Stability of adaptive and nonadaptive packet routing policies in adversarial queueing networks*, SIAM J. Comput., 32 (2003), pp. 371–385.

[14] A. Goel, *Stability of networks and protocols in the adversarial queuing model for packet routing*, Networks, 34 (2001), pp. 219–224.

[15] D. Koukopoulos, S. E. Nikoletseas, and P. G. Spirakis, *The Range of Stability for Heterogeneous and FIFO Queueing Networks*, Electronic Colloquium on Computational Complexity Report TR01-099, 2001. Available online from http://www.eccc.uni-trier.de/eccc.

[16] T. Leighton, B. Maggs, and S. Rao, *Universal packet routing algorithms*, in Proceedings of the 29th Annual Symposium on Foundations of Computer Science, White Plains, NY, 1988, pp. 256–269.

[17] Z. Lotker, B. Patt-Shamir, and A. Rosén, *New stability results for adversarial queuing*, in Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures, Winnipeg, MB, Canada, 2002, pp. 192–199.

[18] Z. Zhang, Z. Duan, and Y. T. Hou, *Fundamental trade-offs in aggregate packet scheduling*, in Proceedings of the 9th Annual IEEE International Conference on Network Protocols, Riverside, CA, 2001.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 304–312

ON IDENTIFYING CODES IN THE TRIANGULAR AND SQUARE GRIDS*

IIRO HONKALA† AND TERO LAIHONEN†

Abstract. It is shown that in the infinite square grid the density of every $(r, \leq 2)$-identifying code is at least $1/8$ and that there exists a sequence C_r of $(r, \leq 2)$-identifying codes such that the density of C_r tends to $1/8$ when $r \to \infty$. In the infinite triangular grid a sequence C'_r of $(r, \leq 2)$-identifying codes is given such that the density of C'_r tends to 0 when $r \to \infty$.

Key words. identifying code, multiprocessor system, triangular lattice, square lattice, density

AMS subject classifications. 94C12, 94B65, 05C70

DOI. 10.1137/S0097539703433110

1. Introduction. Assume that G is an undirected graph and $r > 0$. Denote by V the set of vertices of G. The graphic distance $d(x, y)$ between any two vertices $x, y \in V$ is the number of edges on any shortest path from x to y. A subset $C \subseteq V$ is called an $(r, \leq l)$-identifying code in G if for every subset $F \subseteq V$ of size at most l, the set of codewords (i.e., elements of C) that are within graphic distance r from at least one element in F uniquely identifies F. In other words, if we denote

$$B_r(v) = \{x \in V \mid d(x, v) \leq r\},$$

and

$$I_r(v) = B_r(v) \cap C$$

for all $v \in V$, and

$$I_r(F) = \bigcup_{v \in F} I_r(v)$$

for all $F \subseteq V$, then C is $(r, \leq l)$-identifying in G if the sets $I_r(F)$ are different for all F of size at most l. In particular, since $I_r(\emptyset) = \emptyset$, this implies that $I_r(F) \neq \emptyset$ whenever $F \neq \emptyset$. The set $I_r(F)$ is called the *I-set* of F.

Such codes were introduced in [15]. The motivation is that G represents a multiprocessor architecture: each vertex corresponds to a processor and each edge to a dedicated link between two processors. When maintaining the system, some processors (the codewords) are assigned the task of testing their r-neighbors, and each of them sends the central controller one bit of information telling whether it detected any problems or not. Based on the complete set of answers, the central controller should be able to identify the problem points under the assumption that there are at most l of them. The requirement is clearly that the codewords should form an $(r, \leq l)$-identifying code.

*Received by the editors August 12, 2003; accepted for publication (in revised form) October 13, 2003; published electronically January 22, 2004.
http://www.siam.org/journals/sicomp/33-2/43311.html
†Department of Mathematics, University of Turku, 20014 Turku, Finland (honkala@utu.fi, terolai@utu.fi). The research of the first author was supported by the Academy of Finland under grants 44002 and 200213. The research of the second author was supported by the Academy of Finland under grant 207303.

The architectures that have been mainly studied are the (infinite) square grid (in which $V = \mathbb{Z}^2$, and two vertices are adjacent if their Euclidean distance is 1), the (infinite) king grid (in which $V = \mathbb{Z}^2$, and two vertices are adjacent if their Euclidean distance is 1 or $\sqrt{2}$), the (infinite) triangular grid T (in which

$$V = \left\{ i(1,0) + j\left(\frac{1}{2}, \frac{\sqrt{3}}{2}\right) \mid i,j \in \mathbb{Z} \right\},$$

and two vertices are adjacent if their Euclidean distance is 1), and finally the hexagonal mesh and binary hypercubes, which will not be considered in this paper.

Denote $R_n = \{(x,y) \in \mathbb{Z}^2 \mid |x| \le n, |y| \le n\}$. In the square and king grids the density D of a code C is defined as

$$D = \limsup_{n\to\infty} \frac{|C \cap R_n|}{|R_n|}.$$

Quite a lot is known about how small the density D of an $(r, \le l)$-identifying code can be in the king grid, as summarized in the following table:

King grid			
	$l = 1$	$l = 2$	$l \ge 3$
$r = 1$	$\frac{2}{9}$ [7, 3]	$\frac{9}{22} \le D \le \frac{3}{7}$ [11]	do not exist [11]
$r = 2$	$\frac{1}{4r}$ [2]	$\frac{31}{120} \le D \le \frac{3}{10}$ [11]	do not exist [11]
$r \ge 3$	$\frac{1}{4r}$ [2]	$\frac{1}{4}$ [11]	do not exist [11]

An entry like $\frac{2}{9}$ means that there is such an identifying code with density $\frac{2}{9}$, and that the density of any such code is at least $\frac{2}{9}$; the entry $\frac{9}{22} \le D \le \frac{3}{7}$ means that the density must be at least $\frac{9}{22}$ and that there exists a code with density $D = \frac{3}{7}$.

The purpose of this paper is to consider the triangular and square grids in the case $l = 2$ and $r \ge 2$.

In the triangular grid we denote

$$v(i,j) = i(1,0) + j\left(\frac{1}{2}, \frac{\sqrt{3}}{2}\right)$$

and

$$T_n = \{v(i,j) \mid |i| \le n, |j| \le n\},$$

and the density D of a code C is

$$D = \limsup_{n\to\infty} \frac{|C \cap T_n|}{|T_n|}.$$

For the triangular grid, the corresponding table reads as follows:

Triangular grid			
	$l = 1$	$l = 2$	$l \ge 3$
$r = 1$	$\frac{1}{4}$ [15]	$\frac{9}{16}$ [12]	do not exist [12]
$r \ge 2$	$\frac{2}{6r+3} \le D \le \begin{cases} \frac{1}{2r+4} & \text{if } 4 \mid r \\ \frac{1}{2r+2} & \text{otherwise} \end{cases}$ [1]	$D \to 0$ when $r \to \infty$	do not exist [12]

For $l = 1$, $r = 3$, $D \le \frac{2}{17}$ and for $l = 1$, $r = 5$, $D \le \frac{1}{13}$; see [3].

By the entry $D \to 0$, when $r \to \infty$, we mean that there is a sequence of $(r, \leq 2)$-identifying codes with densities D_r such that $D_r \to 0$ when $r \to \infty$ (see Theorem 1).

Finally, we have the corresponding table for the square grid:

Square grid			
	$l = 1$	$l = 2$	$l \geq 3$
$r = 1$	$\frac{15}{43} \leq D \leq \frac{7}{20}$ [4, 5, 8]	$\frac{1}{2}$ [10]	1 if $l = 3$ [10] do not exist if $l \geq 4$ [10, 16]
$r \geq 2$	$\frac{3}{8r+4} \leq D \leq \begin{cases} \frac{2}{5r} & \text{if } 2 \mid r \\ \frac{2r}{5r^2-2r+1} & \text{if } 2 \not\mid r \end{cases}$ [1, 14]	$D \to \frac{1}{8}$ when $r \to \infty$	do not exist

For $l = 1$, $r = 2$, $D \leq \frac{5}{29}$ by [14];

for $l = 1$, $r = 3, 4, 5, 6$, we have the upper bounds $\frac{1}{8}$, $\frac{8}{85}$, $\frac{2}{25}$, $\frac{3}{46}$ from [3].

Here the entry $D \to \frac{1}{8}$ when $r \to \infty$ comes from Theorems 3 and 5; see also Theorem 6. To be precise, what we prove is that for all r, the density of an $(r, \leq 2)$-identifying code in the square grid must be at least $\frac{1}{8}$, and that there is a sequence of $(r, \leq 2)$-identifying codes with densities D_r such that $D_r \to \frac{1}{8}$ when $r \to \infty$.

For results in the hexagonal mesh, we refer to [1, 3, 6, 8, 13, 15].

In a closely related problem, the requirement is that the sets $I_r(v)$ for $v \in V \setminus C$ are nonempty and no two are the same set; such a code is called r-locating-dominating. For results on such codes, see, e.g., [9, 17, 18].

2. On the triangular grid. The following theorem shows that in the triangular grid we have a sequence of $(r, \leq 2)$-identifying codes with densities D_r such that $D_r \to 0$ when $r \to \infty$.

THEOREM 1. *In the triangular grid there is an $(r, \leq 2)$-identifying code with density $\frac{3r+1}{(2r+1)(r+1)}$.*

Proof. Denote

$$L_k = \{v(i, k(r+1)) \mid i \in \mathbb{Z}\}$$

and

$$C_1 = \bigcup_{k \in \mathbb{Z}} L_k.$$

Denote further

$$C_2 = \{v(i, j) \mid i \equiv 0 \bmod 2r+1\}.$$

We claim that $C_1 \cup C_2$ is $(r, \leq 2)$-identifying. It is easy to check that it has the required density.

We first make two preliminary observations: if $v = v(a, k(r+1)+h)$, then

- $I_r(v) \cap L_k$ is trivially empty if $|h| > r$, and $I_r(v) \cap L_k$ has cardinality $2r+1-|h|$ if $|h| \leq r$, and
- the set $I_r(v) \cap L_k$ is symmetric with respect to the vertical line passing through v.

Assume that $F \subseteq V$ has cardinality at most two and is unknown to us, but that we know $I_r(F)$.

Obviously $I_r(F)$ is empty if and only if F is empty.

Assume first that $I_r(F)$ contains elements from at least three different lines L_k; the lines L_{high} and L_{low} being the highest and lowest among them. Then $|F| = 2$,

and only the vertex v_{high} in F with higher y-coordinate affects $I_r(F) \cap L_{high}$. The two observations made above now immediately tell us the x- and y-coordinates of v_{high}. In the same way we find the other vertex in F by considering $I_r(F) \cap L_{low}$.

Assume then that $I_r(F)$ contains only vertices from one of the lines L_k, say L_s. Then $F \subseteq L_s$. If $v(i_{min}, s(r+1))$ and $v(i_{max}, s(r+1))$ are the vertices in $I_r(F) \cap L_s$ with the smallest and largest x-coordinates, then F consists of the points $v(i_{min} + r, s(r+1))$ and $v(i_{max} - r, s(r+1))$ (which may coincide).

Assume finally that $I_r(F)$ contains elements from exactly two different lines L_k. If they are L_s and L_{s+j} for some $j \geq 2$, then F consists of the middle point in $I_r(F) \cap L_s$ and the middle point in $I_r(F) \cap L_{s+j}$. So assume that they are L_s and L_{s+1}. Then $F \subseteq A$, where $A = \{v(i,j) \mid s(r+1) \leq j \leq (s+1)(r+1)\}$.

We now define four lines H_1, H_2, K_1, and K_2 as follows:

- By considering the leftmost point in $I_r(F) \cap L_{s+1}$, we find a line H_1 with slope $\sqrt{3}$ such that $H_1 \cap A$ must contain a point in F—and the other point in F (if there is one) belongs to $H_1 \cup L_s$ or lies to the right of H_1.
- By considering the rightmost point in $I_r(F) \cap L_s$, we find a line H_2 with slope $\sqrt{3}$ such that $H_2 \cap A$ must contain a point in F—and the other point in F (if there is one) belongs to $H_2 \cup L_{s+1}$ or lies to the left of H_2.
- By considering the leftmost point in $I_r(F) \cap L_s$, we find a line K_1 with slope $-\sqrt{3}$ such that $K_1 \cap A$ must contain a point in F—and the other point in F (if there is one) is in $K_1 \cup L_{s+1}$ or lies to the right of K_1.
- By considering the rightmost point in $I_r(F) \cap L_{s+1}$, we find a line K_2 with slope $-\sqrt{3}$ such that $K_2 \cap A$ must contain a point in F—and the other point in F (if there is one) is in $K_2 \cup L_s$ or lies to the left of K_2.

If $H_1 = H_2$, then by the above description F is a subset of the set consisting of $H_1 \cap A$, the part of L_s which lies to the left of $H_1 \cap L_s$ and of the part of L_{s+1} which lies to the right of $H_1 \cap L_{s+1}$. But due to the structure of this path (and C_1), the points in F are the one obtained by taking r steps to the right along this path from the leftmost codeword in $I_r(F) \cap L_s$, and the one obtained by taking r steps to the left along this path from the rightmost codeword in $I_r(F) \cap L_{s+1}$ (and these two points of F may coincide).

In the same way we find F if $K_1 = K_2$; so assume that $H_1 \neq H_2$ and $K_1 \neq K_2$.

Since H_1, H_2, K_1, and K_2 each contain at least one point of F, we know that either F is the set F_1 consisting of the vertices in $H_1 \cap K_2$ and $H_2 \cap K_1$ (the top and bottom corners of the parallelogram formed by H_1, H_2, K_1, and K_2) or F is the set F_2 consisting of the vertices in $H_1 \cap K_1$ and $H_2 \cap K_2$.

But we can separate between these two remaining cases using C_2. Denote $A_i = \{v(i,j) \mid j \in \mathbb{Z}\}$. Then $H_1 = A_i$ and $H_2 = A_j$ for some $i < j$. By the definition of C_2, there is an index k such that $A_k \subseteq C_2$ and $|k - i| \leq r$. If $k < j$, then the highest point in $I_r(F_1) \cap A_k$ is not in $I_r(F_2)$; if $k \geq j$, then the lowest point in $I_r(F_1) \cap A_k$ is not in $I_r(F_2)$. □

3. On the square grid. We now consider the square grid.

Because always $I_r((-1,-1),(1,1)) = I_r((-1,-1),(0,0),(1,1))$ for $r \geq 2$, there cannot exist an $(r, \leq l)$-identifying code with $r \geq 2$ and $l \geq 3$.

The following theorem gives a useful sufficient condition for a code to be $(r, \leq 2)$-identifying in the square grid.

THEOREM 2. *Assume that $E \subseteq \mathbb{Z}^2$. If every translate of the set*

$$A = \{(-r-1,0),(-r,0),(0,r+1),(0,r),(r+1,0),(r,0),(0,-r-1),(0,-r)\},$$

every translate of the set

$$B = \{(-r, 0), (-r + 1, 1), (-r + 2, 2), \ldots, (0, r)\},$$

and every translate of the set

$$C = \{(0, r), (1, r - 1), (2, r - 2), \ldots, (r, 0)\}$$

contain at least one element of E, then E is $(r, \leq 2)$-identifying in the square grid.

Proof. If $F \neq \emptyset$, say $v \in F$, then $(v + B) \cap E \neq \emptyset$ by the assumption, and hence $I_r(F) \neq \emptyset$.

Assume that F_1 and F_2 are two different nonempty subsets of $\mathbb{Z}^2$ each containing at most two elements and that $I_r(F_1) = I_r(F_2)$. Let R be the smallest rectangle which contains $F_1 \cup F_2$ and whose sides have slopes 1 and -1; possibly R reduces to a line segment.

Assume first that R is a line segment. Assume, for instance, that R has slope -1 and the upper left-hand endpoint v is in F_1 but not F_2 (the other cases are treated similarly). Then $(v + B) \cap E \neq \emptyset$, and any codeword in $(v + B) \cap E$ belongs to $I_r(F_1)$ but not to $I_r(F_2)$.

Assume therefore that R does not reduce to a line segment.

Each side of R must contain a point from both F_1 and F_2. Indeed, assume that, for instance, the upper left-hand side contains a vertex $v \in F_1$ but no point of F_2 (the other cases are similar). By assumption, the set $v + B$ contains a codeword of E, and clearly this codeword belongs to $I_r(F_1)$ but not to $I_r(F_2)$.

But since each side of R contains a point from F_1, and $|F_1| \leq 2$, the only possibility is that F_1 consists of two diagonally opposite corners of R. By the same argument, F_2 must then consist of the other two diagonally opposite corners.

If any of the sides of R contains more than $r + 1$ grid points, say the upper left-hand side (the other cases are similar), and $v \in F_2$, say, is its upper endpoint, then $v + B$ contains a codeword, by assumption, and this codeword is in $I_r(F_2)$ but not in $I_r(F_1)$.

Assume therefore that each side of R contains at most $r + 1$ grid points. Without loss of generality, the leftmost corner of R is the point $(-1, 0)$, and this point belongs to F_1. In particular, this point belongs to $B_{r-1}((-r, 0))$. The bottom corner belongs to F_2 and is one of the points $(0, -1)$, $(1, -2)$, $\ldots$, $(r - 1, -r)$; in particular, it lies in the ball $B_{r-1}((0, -r))$. The top corner of R also belongs to F_2 and is one of the points $(0, 1)$, $(1, 2)$, $\ldots$, $(r - 1, r)$; in particular it lies in the ball $B_{r-1}((0, r))$. Finally the rightmost corner of R must be one of the grid points in the square with the corners $(1, 0)$, $(r, r - 1)$, $(r, -r + 1)$, $(2r - 1, 0)$, so it lies in the ball $B_{r-1}((r, 0))$. Now the claim follows from the fact that A contains at least one codeword. Any two points in A which are not neighbors in the graph are at least graphic distance $2r$ apart. This, the triangle inequality, and what we have just proved show that every element of A is within distance r from a unique corner of R, and any codeword in A therefore belongs to $I_r(F_1) \setminus I_r(F_2)$ or $I_r(F_2) \setminus I_r(F_1)$. □

The following theorem is useful when we want to determine the asymptotic behavior of the density of $(r, \leq 2)$-identifying codes.

THEOREM 3. *For all r, if a code is $(r, \leq 2)$-identifying in the square grid, then its density is at least $\frac{1}{8}$.*

Proof. The set A of Theorem 2 always contains at least one codeword of an $(r, \leq 2)$-identifying code since the I-sets of the sets $\{(-1, 0), (1, 0)\}$ and $\{(0, -1), (0, 1)\}$ (and any of their translates) must be distinct. Because A consists of eight elements,

independently of r, the claim follows by going through all translates of the set A; cf. [1, section 2]. □

For small values of r we get a better lower bound.

THEOREM 4. *If a code is $(r, \leq 2)$-identifying in the square grid, then its density is at least $\frac{1}{(2r+1)}$.*

Proof. The I-sets of $\{(-1,-1)\}$ and $\{(-1,-1),(0,0)\}$ (and any of their translates) must be distinguishable. The symmetric difference of $B_r((-1,-1))$ and $B_r((-1,-1)) \cup B_r((0,0))$ contains $2r+1$ vertices. The claim follows for a fixed r as in the previous proof. □

THEOREM 5. *There is a sequence (D_r) such that $D_r \to \frac{1}{8}$ when $r \to \infty$, and for every r there exists an $(r, \leq 2)$-identifying code in the square grid with density D_r.*

Proof. Denote by H the set of residue classes of 0, 2, 4, ..., $2r$ modulo $4r+1$, and by K the set of residue classes of 0, 1, 2, ..., $2r+1$ modulo $4r+1$.

We claim that the code $C_1 = \{(a,b) \mid a+b \in H, a-b \in K\}$ has the property that every set $(x,y)+A$ (where A is as in Theorem 2) contains at least one codeword; the density of C_1 is clearly $2(r+1)^2/(4r+1)^2$. It is easy to check that for any integer n the set $\{n-r-1, n-r, n+r, n+r+1\}$ contains an element of H. Apply this to $n = x+y$. If $x+y-r-1 \in H$, then at least one of the points $(x-r-1, y)$ and $(x, y-r-1)$ of $(x,y)+A$ belong to C_1, because $x-y-r-1$ or $x-y+r+1$ belongs to K. If $x+y-r \in H$, then $(x-r, y) \in C_1$ or $(x, y-r) \in C_1$; if $x+y+r \in H$, then $(x+r, y) \in C_1$ or $(x, y+r) \in C_1$; if $x+y+r+1 \in H$, then $(x+r+1, y) \in C_1$ or $(x, y+r+1) \in C_1$.

To complete the construction, we take $C_2 = \{(a,b) \mid a \equiv 0 \bmod r+1\}$. Then $C_1 \cup C_2$ clearly satisfies the conditions of Theorem 2, and is hence $(r, \leq 2)$-identifying, and its density tends to $\frac{1}{8}$, when r tends to infinity. □

The previous proof was quite simple, because we did not pay too much attention to the density of the resulting codes.

THEOREM 6. *Let $r \equiv 2 \bmod 8$, $r \geq 18$. Then there exists an $(r, \leq 2)$-identifying code in the square grid with density $\frac{r+1}{8r+4}$.*

Proof. Suppose $r \equiv 2 \bmod 8$ and $r \geq 18$. Denote (see Figure 1 for the case $r = 18$)

$$S = \{(-r-1, 0) + k(8,8) \mid k \in \mathbb{Z}\}$$

and further

$$F = \bigcup_{i \in \mathbb{Z}} (S + i(2,1)).$$

In what follows, by a line with slopes 1 and -1 we always mean a diagonal which intersects with $\mathbb{Z}^2$.

Obviously, every eighth point in a line with slope 1 belongs to F. Also in any line with slope -1 every eighth point is in F. Indeed, if $f \in F$, then $f + q(8,-8) \in F$ for $q \in \mathbb{Z}$ (also $f + q(2,-3) \in F$ and thus every diagonal with slope -1 has an element of F).

Let

$$E_0 = \left(\bigcup_{i \in \{0,1,\ldots,2r\}} (S + i(2,1)) \right) \cup \{(r-1+3+8k, 3+8k) \mid k \in \mathbb{Z}\}.$$

Thus E_0 consists of a part of F with some additional points on the diagonal $(r-1, 0) + (a,a)$, $a \in \mathbb{Z}$.

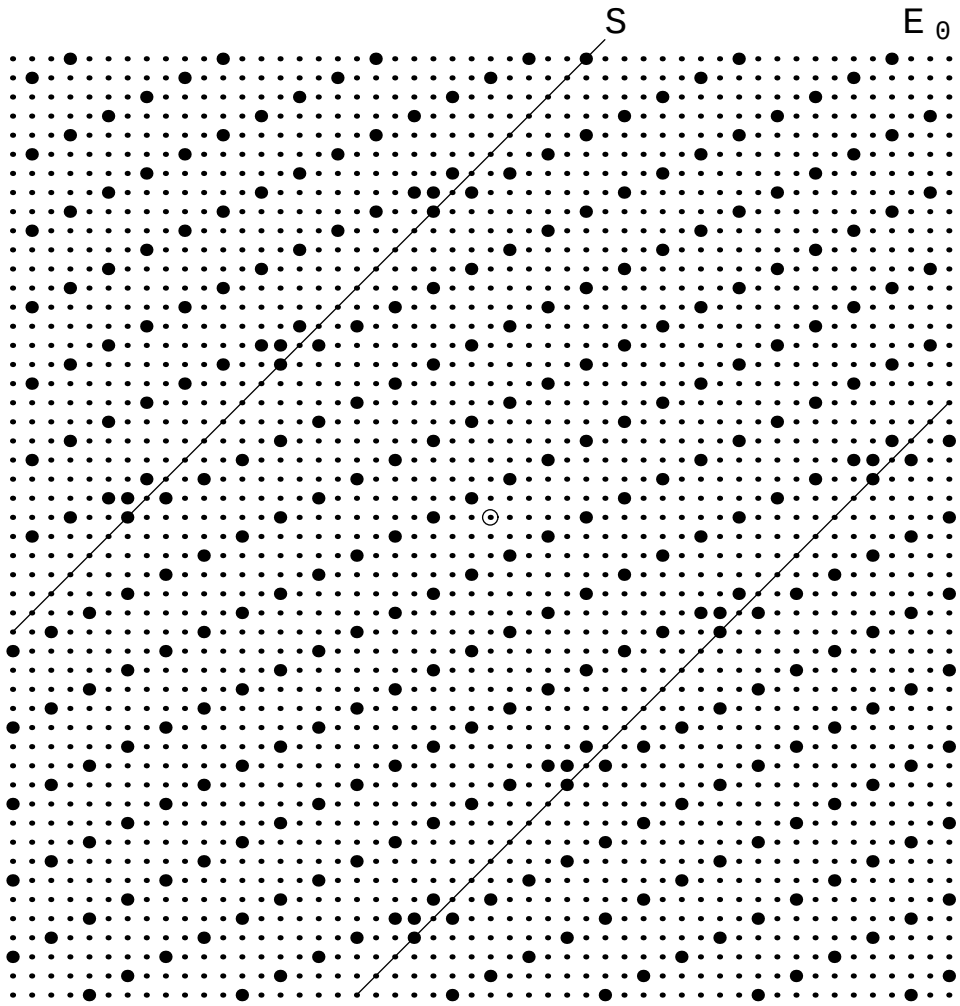


FIG. 1. *A part of the code E for $r = 18$. Codewords are the large black circles and the circled dot is the origin.*

We define the code

$$E = E_r = \bigcup_{j \in \mathbb{Z}} (E_0 + j(2r + 1 + 2, 2)).$$

Hence E consists of translates of E_0 shifted diagonally up by $j(2, 2)$ and horizontally by $j(2r + 1, 0)$.

Observe that E_0 is periodic with period $(8, 8)$ and hence the density of E equals $\frac{2r+2}{8(2r+1)} = \frac{r+1}{8r+4}$.

In what follows we show that E is $(r, \leq 2)$-identifying. This is done by checking the conditions of Theorem 2. We will use the same notation in this theorem for the sets A, B, and C.

Every translate of B contains trivially at least one codeword of E, since at least every eighth point in a diagonal with slope 1 belongs to E.

Every translate of C also contains at least one codeword. Indeed, there can be at most 14 consecutive noncodewords in a line with slope -1. This follows since in $E_0 \cap F$ there can be a gap of at most seven noncodewords and in its translate $E_0 + (2r+1+2, 2)$ there can be another seven. Because $|C| = r+1 \geq 19$, there must be a codeword in every translate of C.

Consider an arbitrary translate $(s+a, a) + A$.

The part

$$A_1 = (s+a, a) + \{(-r-1, 0), (-r, 0), (0, r), (0, r+1)\}$$

of $(s+a, a) + A$ consists of points on two diagonals $(s-r-1, 0) + (b, b)$ and $(s-r, 0) + (b, b)$, $b \in \mathbb{Z}$, and the other half

$$A_2 = \{(s+r+1, 0) + (a, a), (s+r+1, 0) + (a-r-1, a-r-1)\}$$

$$\cup\{(s+r, 0) + (a, a), (s+r, 0) + (a-r, a-r)\}$$

of $(s+a, a) + A$ consists of points on two other diagonals, which are the same two diagonals shifted diagonally up by $(2, 2)$ and horizontally by $(2r+1, 0)$. Shifting back, we know that A_2 contains at least one codeword if and only if the set

$$A_3 := \{(s-r, 0) + (a-2, a-2), (s-r, 0) + (a-r-3, a-r-3)\}$$

$$\cup\{(s-r-1, 0) + (a-2, a-2), (s-r-1, 0) + (a-r-2, a-r-2)\}$$

does. We now consider $A_1 \cup A_3$ instead.

If $s \not\equiv 2r \bmod 2r+1$, then our two diagonals are from the same translate of E_0. If none of the four points of $A_1 \cup A_3$ in

$$(s-r, 0) + \{(a, a), (a+r, a+r), (a-2, a-2), (a-r-3, a-r-3)\}$$

is in the code, then (shifting back by $(2, 1)$) none of the points in

$$(s-r-1, 0) + \{(a-1, a-1), (a+r-1, a+r-1), (a-3, a-3), (a-r-4, a-r-4)\}$$

is in the code either. But then at least one of the points of $A_1 \cup A_3$ in

(1)
$$(s-r-1, 0) + \{(a, a), (a+r+1, a+r+1), (a-2, a-2), (a-r-2, a-r-2)\}$$

must be in the code, because

$$a, a+r+1, a-2, a-r-2, a-1, a+r-1, a-3, a-r-4$$

belong to distinct residue classes modulo eight when $r \equiv 2 \bmod 8$.

If $s \equiv 2r \bmod 2r+1$, then our two diagonals are the rightmost diagonal in a translate of E_0 and the leftmost diagonal in the next translate of E_0. Choose j so that $s-r-1 = r-1+j(2r+1)$. Then E contains all the points $(s-r-1, 0) + (b, b)$, where $b \equiv 3+2j \bmod 8$ or $b \equiv 4+2j \bmod 8$; and in particular, for all $b \equiv 4+2j \bmod 8$ also the point obtained by shifting by $-2r(2, 1)$ is in E. From (1) and the fact that the residue classes of a, $a+r+1$, $a-2$, and $a-r-2$ are a, $a+3$, $a+4$, and $a+6$, we see that the only way none of the points in (1) would be in the code is if $3+2j \equiv a+1 \bmod 8$ and $4+2j \equiv a+2 \bmod 8$, and then $(s-r-1, 0) + (a+2r-2, a+2r-2) \in E$, and hence $v = (s-r-1, 0) + (a+2r-2, a+2r-2) - 2r(2, 1) \in E$ (as we observed earlier), and finally $(s-r, 0) + (a, a) = v + (2r+1+2, 2) \in E$, which proves the claim, because this point is in $A_1 \cup A_3$. □

REFERENCES

[1] I. Charon, I. Honkala, O. Hudry, and A. Lobstein, *General bounds for identifying codes in some infinite regular graphs*, Electron. J. Combin., 8 (2001), Research Paper 39.

[2] I. Charon, I. Honkala, O. Hudry, and A. Lobstein, *The minimum density of an identifying code in the king lattice*, Discrete Math., to appear.

[3] I. Charon, O. Hudry, and A. Lobstein, *Identifying codes with small radius in some infinite regular graphs*, Electron. J. Combin., 9 (2002), Research Paper 11.

[4] G. Cohen, S. Gravier, I. Honkala, A. Lobstein, M. Mollard, C. Payan, and G. Zémor, *Improved identifying codes for the grid*, Electron. J. Combin., 6 (1999), Research Paper 19, comment.

[5] G. Cohen, I. Honkala, A. Lobstein, and G. Zémor, *New bounds for codes identifying vertices in graphs*, Electron. J. Combin., 6 (1999), Research Paper 19.

[6] G. D. Cohen, I. Honkala, A. Lobstein, and G. Zémor, *Bounds for codes identifying vertices in the hexagonal grid*, SIAM J. Discrete Math., 13 (2000), pp. 492–504.

[7] G. Cohen, I. Honkala, A. Lobstein, and G. Zémor, *On codes identifying vertices in the two-dimensional square lattice with diagonals*, IEEE Trans. Comput., 50 (2001), pp. 174–176.

[8] G. Cohen, I. Honkala, A. Lobstein, and G. Zémor, *On identifying codes*, in Codes and Association Schemes, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 56, AMS, Providence, RI, 2001, pp. 97–109.

[9] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, *Fundamentals of Domination in Graphs*, Marcel Dekker, New York, 1998.

[10] I. Honkala and T. Laihonen, *On the identification of sets of points in the square lattice*, Discrete Comput. Geom., 29 (2003), pp. 139–152.

[11] I. Honkala and T. Laihonen, *Codes for identification in the king lattice*, Graphs Combin., to appear.

[12] I. Honkala and T. Laihonen, *On identification in the triangular lattice*, J. Combin. Theory Ser. B, submitted.

[13] I. Honkala and T. Laihonen, *On identifying codes in the hexagonal mesh*, Inform. Process. Lett., to appear.

[14] I. Honkala and A. Lobstein, *On the density of identifying codes in the square lattice*, J. Combin. Theory Ser. B, 85 (2002), pp. 297–306.

[15] M. G. Karpovsky, K. Chakrabarty, and L. B. Levitin, *On a new class of codes for identifying vertices in graphs*, IEEE Trans. Inform. Theory, 44 (1998), pp. 599–611.

[16] T. Laihonen and S. Ranto, *Codes identifying sets of vertices*, in Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, Lecture Notes in Comput. Sci. 2227, Springer-Verlag, Berlin, 2001, pp. 82–91.

[17] P. J. Slater, *Locating dominating sets and locating-dominating sets*, in Graph Theory, Combinatorics and Applications: Proceedings of the Seventh Quadrennial International Conference on the Theory and Applications of Graphs, Y. Alavi and A. Schwenk, eds., Wiley, New York, 1995, pp. 1073–1079.

[18] P. J. Slater, *Fault-tolerant locating-dominating sets*, Discrete Math., 249 (2002), pp. 179–189.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 313–331

A BOUND ON THE CAPACITY OF BACKOFF AND ACKNOWLEDGMENT-BASED PROTOCOLS*

LESLIE ANN GOLDBERG†, MARK JERRUM‡, SAMPATH KANNAN§, AND MIKE PATERSON†

Abstract. We study contention-resolution protocols for multiple-access channels. We show that *every* backoff protocol is transient if the arrival rate, λ, is at least 0.42 and that the capacity of every backoff protocol is at most 0.42. Thus, we show that backoff protocols have (provably) smaller capacity than full-sensing protocols. Finally, we show that the corresponding results, with the larger arrival bound of 0.531, also hold for every acknowledgment-based protocol.

Key words. contention-resolution, backoff, multiple-access channel, stability, protocol

AMS subject classifications. 68W20, 68W15

DOI. 10.1137/S0097539700381851

1. Introduction.

A *multiple-access channel* is a broadcast channel that allows multiple users to communicate with each other by sending messages onto the channel. If two or more users simultaneously send messages, then the messages interfere with each other (collide), and the messages are not transmitted successfully. The channel is not centrally controlled. Instead, the users use a contention-resolution protocol to resolve collisions. Thus, after a collision, each user involved in the collision waits a random amount of time (which is determined by the protocol) before resending.

Following previous work on multiple-access channels, we work in a *time-slotted* model in which time is partitioned into discrete *time steps*. At the beginning of each time step, a random number of messages enter the system, each of which is associated with a new user which has no other messages to send. The number of messages that enter the system is drawn from a Poisson distribution with mean λ. During each time step, each message chooses independently whether to send to the channel. If *exactly* one message sends to the channel during the time step, then this message leaves the system and we call this a *success*. Otherwise, all of the messages remain in the system and the next time step is started. Note that when a message sends to the channel this may or may not result in a success, depending on whether any other messages send to the channel.

The quality of a protocol can be measured in several ways. Typically, one models the execution of the protocol as a Markov chain. If the protocol is good (for a given arrival rate λ), the corresponding Markov chain will be *recurrent* (with probability 1, it will eventually return to the empty state in which no messages are waiting). Otherwise, the chain is said to be *transient* (and we also say that a protocol is tran-

*Received by the editors December 1, 2000; accepted for publication (in revised form) October 7, 2003; published electronically January 22, 2004. This work was partially supported by the EPSRC grant "Sharper Analysis of Randomised Algorithms: A Computational Approach," NSF grant CCR9820885, and the IST Programme of the EU under contracts IST-1999-14186 (ALCOM-FT) and IST-1999-14036 (RAND-APX).

http://www.siam.org/journals/sicomp/33-2/38185.html

†Department of Computer Science, University of Warwick, Coventry, CV4 7AL, United Kingdom (http://www.dcs.warwick.ac.uk/∼leslie/; http://www.dcs.warwick.ac.uk/∼msp/).

‡Division of Informatics, University of Edinburgh, JCMB, The King's Buildings, Edinburgh EH9 3JZ, United Kingdom (http://www.dcs.ed.ac.uk/∼mrj/).

§Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd St., Philadelphia, PA 19104-6389 (http://www.cis.upenn.edu/∼kannan/).

sient). Note that transience is a very strong form of instability. In particular, if we focus on any finite set of "good" states, then if the chain is transient, the probability of visiting these states at least N times during the infinite run of the protocol is exponentially small in N. (This follows because the relevant Markov chain is irreducible and aperiodic.)

Another way to measure the quality of a protocol is to measure its *capacity*. A protocol is said to achieve *full throughput* at rate λ if, when it is run with input rate λ, the average success rate is λ. The *capacity* of the protocol [4] is the maximum arrival rate at which it achieves full throughput.

The protocols that we consider in this paper are *acknowledgment-based* protocols. In the acknowledgment-based model, the only information that a user receives about the state of the system is the history of its own transmissions. An alternative model is the *full-sensing* model, in which *every* user listens to the channel at *every* step, regardless of whether it sends during the step.[1]

One particularly simple and easy-to-implement class of acknowledgment-based protocols is the class of *backoff protocols*. A backoff protocol is a sequence of probabilities $p_0, p_1, \ldots$. If a message has sent unsuccessfully i times before a time step, then, with probability p_i, it sends during the time step. Otherwise, it does not send. Kelly and MacPhee [13, 14, 17] gave a formula for the *critical arrival rate*, λ^*, of a backoff protocol, which is the minimum arrival rate for which the expected number of successful transmissions that the protocol makes is finite.[2]

Perhaps the best-known backoff protocol is the *binary exponential backoff protocol* in which $p_i = 2^{-i}$. This protocol is the basis of the Ethernet protocol of Metcalfe and Boggs [18].[3] Kelly and MacPhee showed that the critical arrival rate of this protocol is $\ln 2$. Thus, if $\lambda > \ln 2$, then binary exponential backoff achieves only a finite number of successful transmissions (in expectation). Aldous [1] showed that the binary exponential backoff protocol is not a good protocol for *any* positive arrival rate λ. In particular, it is transient and the expected number of successful transmissions in t steps is $o(t)$. MacPhee [17] posed the question of whether there exists a backoff protocol which is recurrent for some positive arrival rate λ.

In this paper, we show that there is no backoff protocol which is recurrent for $\lambda \geq 0.42$. (Thus, *every* backoff protocol is transient if $\lambda \geq 0.42$.) Also, every backoff protocol has capacity at most 0.42. As far as we know, our result is the first proof showing that backoff protocols have smaller capacity than full-sensing protocols. In particular, Mosely and Humblet [20] have discovered a full-sensing protocol with capacity 0.48776.[4] Finally, we show that *no* acknowledgment-based protocol is recurrent for $\lambda \geq 0.530045$.

[1] In practice, it is possible to implement the full-sensing model when there is a single channel, but this becomes increasingly difficult in situations where there are multiple shared channels, such as optical networks. Thus, acknowledgment-based protocols are sometimes preferable to full-sensing protocols. For work on contention-resolution in the multiple-channel setting, see [6].

[2] If $\lambda > \lambda^*$, then the expected number of successes is finite, even if the protocol runs forever. They showed that the critical arrival rate is 0 if the expected number of times that a message sends during the first t steps is $\omega(\log t)$.

[3] There are several differences between the "real-life" Ethernet protocol and "pure" binary exponential backoff, but we do not describe these here.

[4] Mosely and Humblet's protocol is a "tree protocol" in the sense of Capetanakis [3] and Tsybakov and Mikhailov [25]. For a simple analysis of the protocol, see [26]. Vvedenskaya and Pinsker have shown how to modify Mosely and Humblet's protocol to achieve an improvement in the capacity (in the seventh decimal place) [27].

1.1. Related work. Backoff protocols and acknowledgment-based protocols have also been studied in an n-user model, which combines contention-resolution with queueing. In this model, it is assumed that n users maintain queues of messages, and that new messages arrive at the tails of the queues. At each step, the users use contention-resolution protocols to try to send the messages at the heads of their queues. It turns out that the queues have a stabilizing effect, so some protocols (such as "polynomial backoff") which are unstable in our model [14] are stable in the queueing model [12]. We will not describe queueing-model results here but refer the reader to [2, 9, 12, 22].

Much work has gone into determining upper bounds on the capacity that can be achieved by a full-sensing protocol. The current best result is due to Tsybakov and Likhanov [24], who have shown that no protocol can achieve capacity higher than 0.568. (For more information, see [4, 10, 19, 23].) In the full-sensing model, one typically assumes that messages are born at real "times" which are chosen uniformly from the unit interval. Recently, Loher [15, 16] has shown that if a protocol is required to respect these birth times, in the sense that packets must be successfully delivered in their birth order, then no protocol can achieve capacity higher than 0.4906. Intuitively, the "first-come-first-served" restriction seems very strong, so it is somewhat surprising that the best-known algorithm without the restriction (that of Vvedenskaya and Pinsker) does not beat this upper bound. The algorithm of Humblet and Mosely satisfies the first-come-first-served restriction.

2. Markov chain background. A Markov chain $X = \{X_0, X_1, \ldots\}$ with a countable state space Ω (see [11]) is *time-homogeneous* if its transition probabilities are independent of time so $\Pr(X_{n+1} = j \mid X_n = i) = \Pr(X_1 = j \mid X_0 = i)$ for all n, i, j. It is *irreducible* if every pair (i, j) of states is connected in the sense that there is an $n > 0$ such that $\Pr(X_{n+m} = j \mid X_m = i) > 0$. It is *aperiodic* if every state i satisfies $\gcd\{n \mid \Pr(X_{n+m} = i \mid X_m = i) > 0\} = 1$. If the chain is irreducible and aperiodic, then we say that it is *recurrent* if it returns to its start state with probability 1. That is, it is recurrent if for some state i (and therefore for all i), $\text{Prob}[X_t = i \text{ for some } t \geq 1 \mid X_0 = i] = 1$. Otherwise, X is said to be *transient*. X is *positive recurrent* (or ergodic) if the expected number of steps that it takes before returning to its start state is finite. A chain is positive recurrent if and only if it has a unique stationary distribution. The standard way to prove that a Markov chain is positive recurrent is Foster's theorem.

THEOREM 1 (Foster [7]). *A time-homogeneous irreducible aperiodic Markov chain X with a countable state space Ω is positive recurrent if and only if there exists a positive function $f(\rho)$, $\rho \in \Omega$, a number $\epsilon > 0$, and a finite set $\mathcal{A} \subseteq \Omega$ such that the following inequalities hold:*

$$E[f(X(t+1)) - f(X(t)) \mid X(t) = \rho] \leq -\epsilon, \quad \rho \notin \mathcal{A}, \tag{1}$$

$$E[f(X(t+1)) \mid X(t) = \rho] < \infty, \quad \rho \in \mathcal{A}. \tag{2}$$

Basically, the idea is to use a "potential function" f to follow the progress of the chain. The chain is positive recurrent if and only if there is a potential function f which

1. usually decreases (equation (1)), and
2. cannot increase much (equation (2))

in a single step. Equation (1) implies that, from any state $\rho \notin \mathcal{A}$, the expected time to reach $\mathcal{A}$ from ρ is at most $f(\rho)/\epsilon$. This (combined with (2)) implies that the expected return time to $\mathcal{A}$ is finite, which in turn implies that the chain is positive recurrent.

(For more details, see [5].) Theorems like Theorem 1 are called "drift theorems" because the progress of the Markov chain X is studied by focusing on the "drift" of the potential function f. The function f is sometimes called a *Lyapunov function* or a *test function*.

We can also use drift theorems to show that a Markov chain is *not* positive recurrent. To do this we want to find a potential function f which "drifts" towards larger potentials. Here is the theorem that we will use.

THEOREM 2 (Fayolle, Malyshev, and Menshikov [5]). *An irreducible aperiodic time-homogeneous Markov chain X with countable state space Ω is not positive recurrent if there is a function f with domain Ω and there are constants C and d such that*

1. *there is a state x with $f(x) > C$, and a state y with $f(y) \leq C$,*
2. *$E[f(X_1) - f(X_0) \mid X_0 = x] \geq 0$ for all x with $f(x) > C$, and*
3. *$E[\,|f(X_1) - f(X_0)| \mid X_0 = x] \leq d$ for every state x.*

We will use a similar theorem to show that a Markov chain is transient (which is stronger than saying that it is not positive recurrent).

THEOREM 3 (Fayolle, Malyshev, and Menshikov [5]). *An irreducible aperiodic time-homogeneous Markov chain X with countable state space Ω is transient if there is a positive function f with domain Ω and there are positive constants C, d, and ε such that*

1. *there is a state x with $f(x) > C$, and a state y with $f(y) \leq C$,*
2. *$E[f(X_1) - f(X_0) \mid X_0 = x] \geq \varepsilon$ for all x with $f(x) > C$, and*
3. *if $|f(x) - f(y)| > d$, then the probability of moving from x to y in a single move is 0.*

3. Stochastic domination and monotonicity. Suppose that X is a Markov chain and that the (countable) state space Ω of the chain is a *partial order* with binary relation $\leq$. If A and B are random variables taking states as values, then B *dominates* A if and only if there is a joint sample space for A and B in which the value of A is always less than or equal to the value of B. Note that there will generally be other joint sample spaces in which the value of A can exceed the value of B. We write $A \leq B$ to indicate that B dominates A. We say that X is *monotonic* if for any states $x \leq x'$, the next state conditioned on starting at x' dominates the next state conditioned on starting at x. (Formally, $(X_1 \mid X_0 = x')$ dominates $(X_1 \mid X_0 = x)$.)

When an acknowledgment-based protocol is viewed as a Markov chain, the state is just the collection of messages in the system. (Each message is identified by the history of its transmissions.) Thus, the state space is countable and forms a partial order with respect to the subset inclusion relation $\subseteq$ (for multisets). We say that a protocol is *deletion resilient* [8] if its Markov chain is monotonic with respect to the subset-inclusion partial order.

OBSERVATION 4. *Every acknowledgment-based protocol is deletion resilient.*

Proof. Consider the states x and x' with $x \subseteq x'$. Recall that each state is a set of messages, each message being identified by its transmission history. Thus, x' contains all of the messages in x and possibly others. Now consider one step of the protocol. We wish to show that the random variable denoting the next state $z' = (X_1 \mid X_0 = x')$ dominates the random variable $z = (X_1 \mid X_0 = x)$. z' does dominate z because we can draw z and z' from a joint sample space in which

- the messages in x do the same thing in both copies, and
- both copies have the same number of new arrivals, which make the same number of send attempts in both copies.

Now consider any message m which is either a new arrival or a member of x.

1. If m is silent during the step, then its transmission history in z' is the same as in z.
2. If m has a collision during the transition to z, then it also has a collision during the transition to z', so its transmission history in z' is the same as in z.

Thus, $z \subseteq z'$. □

As we indicated earlier, we will generally assume that the number of messages entering the system at a given step is drawn from a Poisson process with mean λ. However, it will sometimes be useful to consider other message-arrival distributions. If I and I' are message-arrival distributions, we write $I \leq I'$ to indicate that the number of messages generated under I is dominated by the number of messages generated under I'.

OBSERVATION 5. *If the acknowledgment-based protocol P is recurrent under the message-arrival distribution I' and $I \leq I'$, then P is also recurrent under I.*

Proof. Let X be the Markov chain corresponding to protocol P with arrival distribution I with X_0 as the empty state. Let X' be the analogous Markov chain with arrival distribution I'. Consider the evolution of the stochastic process (X_0, X'_0), $(X_1, X'_1), \ldots$. We will choose the random variable (X_i, X'_i) from a joint probability distribution in which

1. every message which is common to X_i and X'_i does the same thing in both copies;
2. the new arrivals which are drawn from I arrive in both copies and make the same number of send attempts in both copies;
3. some additional messages may arrive in X'_i (according to I').

Note that items (2) and (3) are possible since $I \leq I'$. We can now show by induction on t that X'_t dominates X_t. That is, when this joint distribution is used, X_t is a subset of X'_t. This holds for $t = 0$ since $X_0 = X'_0$. The inductive step is the same as in the proof of Observation 4. Consider any message m which is in X_t or arrives (according to I) just before step $t + 1$.

1. If m is silent during the step, then its transmission history in X'_{t+1} is the same as in X_{t+1}.
2. If m has a collision during the transition to X_{t+1}, then it also has a collision during the transition to X'_{t+1}.

Thus, $X_{t+1} \subseteq X'_{t+1}$. Finally, since X'_t dominates X_t, the recurrence of X'_t implies the recurrence of X_t. □

4. Backoff protocols. In this section, we will show that there is no backoff protocol which is recurrent for $\lambda \geq 0.42$. Our method will be to use the drift theorems in section 2. Let $p_0, p_1, \ldots$ be a backoff protocol. Without loss of generality, we can assume $p_0 = 1$, since we can ignore new arrivals until they first send.[5] Let $\lambda = 0.42$. Let X be the Markov chain described in section 3 which describes the behavior of the protocol with arrival rate λ. First, we will construct a potential function (Lyapunov function) f which satisfies the conditions of Theorem 2, that is, a potential function which has a bounded positive drift. We will use Theorem 2 to conclude that the chain is not positive recurrent. Next, we will consider the behavior of the protocol under a *truncated* arrival distribution, and we will use Theorem 3 to show that the protocol

[5] Since the arrivals are Poisson, and Poisson random variables are additive, the number of messages making their very first send on a given time step is Poisson, and the mean of this distribution approaches λ.

is transient. Using Observation 5 (domination), we will conclude that the protocol is also transient with Poisson arrivals at rate λ or higher. Finally, we will show that the *capacity* of every backoff protocol is at most 0.42.

We will use the following technical lemma.

LEMMA 6. *Let* $1 \le t_i \le d$ *for* $i \in [1,k]$ *and* $\prod_{i=1}^{k} t_i = c$. *Then* $\sum_{i=1}^{k}(t_i - 1) \le (d-1)\frac{\log c}{\log d}$.

Proof. Let $S = \sum_{i=1}^{k} t_i$. S can be viewed as a function of $k-1$ of the t_i's; for example, $S = \sum_{i=1}^{k-1} t_i + c/\prod_{i=1}^{k-1} t_i$. For $i \in \{1,\ldots,k-1\}$, the derivative of S with respect to t_i is $1 - c/(t_i \prod_{j=1}^{k-1} t_j)$. Thus, the derivative is positive if $t_i > t_k$. Thus, S is maximized (subject to c) by setting some t_i's to 1, some t_i's to d, and at most one t_i to some intermediate value $t \in [1,d)$. Given this, the maximum value of $\sum_{i=1}^{k}(t_i - 1)$ is $s(d-1) + t - 1$, where $c = d^s t$ and $s = \lfloor (\log c)/(\log d) \rfloor$. Let α be the fractional part of $(\log c)/(\log d)$, that is, $\alpha = (\log c)/(\log d) - s$. We want to show that $s(d-1) + t - 1 \le (d-1)(\log c)/(\log d)$. This is true, since

$$\begin{aligned}(d-1)\frac{\log c}{\log d} - s(d-1) - (t-1) &= \alpha(d-1) - c/d^s + 1 \\ &= \alpha(d-1) - d^\alpha + 1 \\ &\ge 0.\end{aligned}$$

The final inequality holds since we have equality for $d = 1$, and the partial derivative with respect to d proves that the inequality holds for $d > 1$. □

We now define some parameters of a state x. Let $k(x)$ denote the number of messages in state x. If $k(x) = 0$, then $p(x) = r(x) = u(x) = 0$. Otherwise, let $m_1,\ldots,m_{k(x)}$ denote the messages in state x, with send probabilities $\rho_1 \ge \cdots \ge \rho_{k(x)}$. Let $p(x) = \rho_1$ and let $r(x)$ denote the probability that at least one of $m_2,\ldots,m_{k(x)}$ sends on the next step. Let $u(x)$ denote the probability that exactly one of $m_2,\ldots,m_{k(x)}$ sends on the next step. Clearly $u(x) \le r(x)$. If $p(x) < r(x)$, then we use the following (tighter) upper bound for $u(x)$.

LEMMA 7. *If* $p(x) < r(x)$, *then* $u(x) \le \frac{r(x) - r(x)^2/2}{1 - p(x)/2}$.

Proof. Fix a state x. We will use $k, p, r, \ldots$ to denote $k(x), p(x), r(x), \ldots$. Since $p < r$, we have $k \ge 2$.

$$u = \sum_{i=2}^{k} \frac{\rho_i}{1-\rho_i} \prod_{i=2}^{k}(1-\rho_i) = \sum_{i=2}^{k}(t_i - 1)(1-r),$$

where $t_i = 1/(1-\rho_i)$. Let $d = 1/(1-p)$, and note that $1 \le t_i \le d$. By Lemma 6

$$\begin{aligned}u \le (1-r)(d-1)\frac{\log(\prod_{i=2}^{k} t_i)}{\log d} &= (1-r)\frac{p}{1-p}\frac{\log(1/(1-r))}{\log d} \\ &= (1-r)\frac{p}{1-p}\frac{(-\log(1-r))}{(-\log(1-p))}.\end{aligned}$$

Now we wish to show that

$$(1-r)\frac{p}{1-p}\frac{(-\log(1-r))}{(-\log(1-p))} \le \frac{r - r^2/2}{1 - p/2},$$

i.e., that

$$(1-r)\frac{(-\log(1-r))}{r - r^2/2} \le (1-p)\frac{(-\log(1-p))}{p - p^2/2}.$$

This is true, since the function $(1-r)^{\frac{(-\log(1-r))}{r-r^2/2}}$ is decreasing in r. To see this, note that the derivative of this function with respect to r is $y(r)/(r-r^2/2)^2$, where

$$\begin{aligned} y(r) &= (1-r+r^2/2)\log(1-r) + (r-r^2/2) \\ &\leq (1-r+r^2/2)(-r-r^2/2) + (r-r^2/2) = -r^4/4. \end{aligned}$$ □

Let $\mathcal{S}(x)$ denote the probability that there is a success when the system is run for one step starting in state x. (Recall that a success occurs if *exactly* one message sends during the step. This single sender might be a new arrival, or it might be an old message from state x.) Let

$$g(r,p) = e^{-\lambda}\left[(1-r)p + (1-p)\min\left\{r, \frac{r-r^2/2}{1-p/2}\right\} + (1-p)(1-r)\lambda\right].$$

We now have the following corollary of Lemma 7.

COROLLARY 8. *For any state x, $\mathcal{S}(x) \leq g(r(x),p(x))$.*

Let $s(x)$ denote the probability that at least one message in state x sends on the next step. That is, $s(x)$ is the probability that at least one *existing* message in x sends. New arrivals may also send. There may or may not be a success. (Thus, if x is the empty state, then $s(x) = 0$.) Let $A = 0.9$ and $B = 0.41$. For every $z \in [0,1]$, let $c(z) = \max(0, -A\,z + B)$. For every state x, let $f(x) = k(x) + c(s(x))$. The function f is the potential function alluded to earlier, which plays a leading role in Theorems 2 and 3. To a first approximation, $f(x)$ counts the number of messages in the state x, but the small correction term is crucial. Finally, let

$$h(r,p) = \lambda - g(r,p) - [1 - e^{-\lambda}(1-p)(1-r)(1+\lambda)]c(r+p-r\,p) + e^{-\lambda}p(1-r)c(r).$$

Now we have the following.

OBSERVATION 9. *For any state x, $E[\,|f(X_1) - f(X_0)|\mid X_0 = x] \leq 1 + B$.*

LEMMA 10. *For any state x, $E[f(X_1) - f(X_0) \mid X_0 = x] \geq h(r(x),p(x))$.*

Proof. The result follows from the following chain of inequalities, each link of which is justified below:

$$\begin{aligned} &E[f(X_1) - f(X_0) \mid X_0 = x] \\ &\quad = \lambda - \mathcal{S}(x) + E[c(s(X_1)) \mid X_0 = x] - c(s(x)) \\ &\quad \geq \lambda - g(r(x),p(x)) + E[c(s(X_1)) \mid X_0 = x] - c(s(x)) \\ &\quad \geq \lambda - g(r(x),p(x)) + e^{-\lambda}(1-p(x))(1-r(x))(1+\lambda)c(s(x)) \\ &\qquad\quad + e^{-\lambda}p(x)(1-r(x))c(r(x)) - c(s(x)) \\ &\quad = h(r(x),p(x)). \end{aligned}$$

The first inequality follows from Corollary 8. The second comes from substituting exact expressions for $c(s(X_1))$ whenever the form of X_1 allows it, and using the bound $c(s(X_1)) \geq 0$ elsewhere. If none of the existing messages sends and there is at most one arrival, then $c(s(X_1)) = c(s(x))$, giving the third term; if message m_1 alone sends and there are no new arrivals, then $c(s(X_1)) = c(r(x))$, giving the fourth term. The final equality uses the fact that $s(x) = p(x) + r(x) - p(x)r(x)$. □

LEMMA 11. *For any $r \in [0,1]$ and $p \in [0,1]$, $h(r,p) \geq 0.003$.*

Proof. We defer the proof of this lemma to the appendix. Figure 1 contains a (Mathematica-produced) plot of $-h(r,p)$ over the range $r \in [0,1]$, $p \in [0,1]$. The plot suggests that $-h(r,p)$ is bounded below zero.

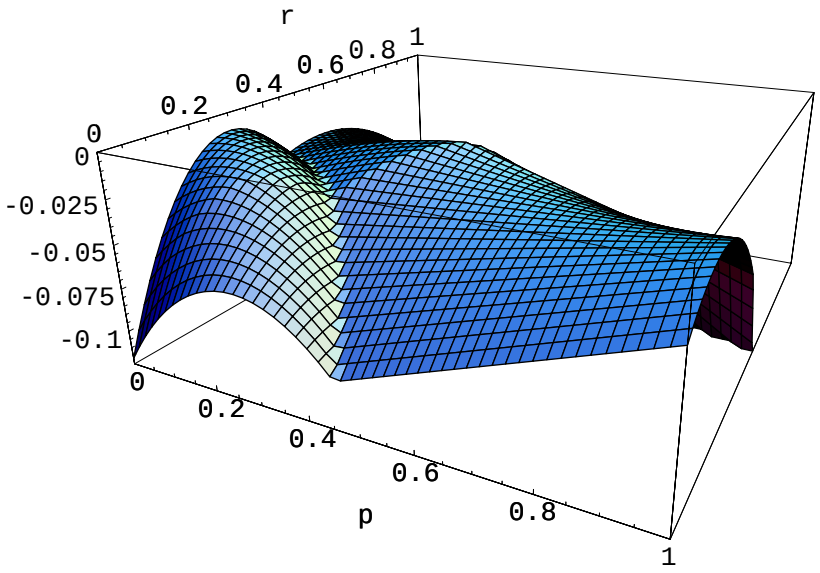


FIG. 1. $-h(r,p)$ *over the range* $r \in [0,1]$, $p \in [0,1]$.

We note here that our proof of the lemma (in the appendix) involves evaluating certain polynomials at about 40,000 points, and we did this using Mathematica. □

We now have the following theorem.

THEOREM 12. *No backoff protocol is positive recurrent when the arrival rate is* $\lambda = 0.42$.

Proof. This follows from Theorem 2, Observation 9, and Lemmas 10 and 11. The value C in Theorem 2 can be taken to be 1 and the value d can be taken to be $1 + B$. □

Now we wish to show that every backoff protocol is transient for $\lambda \geq 0.42$. Once again, fix a backoff protocol $p_0, p_1, \ldots$ with $p_0 = 1$. Notice that our potential function f almost satisfies the conditions in Theorem 3. The main problem is that there is no absolute bound on the amount that f can change in a single step, because the arrivals are drawn from a Poisson distribution. We get around this problem by first considering a *truncated-Poisson* distribution, $T_{M,\lambda}$, in which the probability of r inputs is $e^{-\lambda}\lambda^r/r!$ (as for the Poisson distribution) when $r < M$, but $r = M$ for the remaining probability. By choosing M sufficiently large we can have $E[T_{M,\lambda}]$ arbitrarily close to λ.

LEMMA 13. *Every backoff protocol is transient for the input distribution* $T_{M,\lambda}$ *when* $\lambda = 0.42$ *and* $\lambda' = E[T_{M,\lambda}] > \lambda - 0.001$.

Proof. The proof is almost identical to that of Theorem 12, except that the first term, λ, in the definition of $h(r,p)$ (for Lemmas 10 and 11) must be replaced by λ'. The corresponding function h' satisfies $h'(r,p) \geq h(r,p) - 0.001$. Thus Lemma 11 shows that $h'(r,p) \geq 0.002$ for all $r \in [0,1]$ and $p \in [0,1]$.

The potential function $f(x)$ is defined as before, but under the truncated input distribution we have the property required for Theorem 3. If $|f(x) - f(y)| > M + B$, then the probability of moving from x to y in a single move is 0.

The lemma follows from Theorem 3, where the values of C, ε, and d can be taken to be 1, 0.002, and $M + B$, respectively. □

We now have the following theorem.

THEOREM 14. *Every backoff protocol is transient under the Poisson distribution with arrival rate* $\lambda \geq 0.42$.

Proof. The proof is immediate from Lemma 13 and Observation 5. □

Finally, we bound the capacity of every backoff protocol.

THEOREM 15. *The capacity of every backoff protocol is at most* 0.42.

Proof. Let $p_0, p_1, \ldots$ be a backoff protocol, let $\lambda'' \geq 0.42$ be the arrival rate, and let $\lambda = 0.42$. View the arrivals at each step as Poisson(λ) "ordinary" messages and Poisson($\lambda'' - \lambda$) "ghost" messages. We will show that the protocol does not achieve average success rate λ''. Let $Y_0, Y_1, \ldots$ be the Markov chain describing the protocol. Let $k(Y_t)$ be the number of ordinary messages in the system after t steps. Clearly, the expected number of successes in the first t steps is at most $\lambda'' t - E[k(Y_t)]$. Now let $X_1, X_2, \ldots$ be the Markov chain describing the evolution of the backoff protocol with arrival rate λ (with no ghost messages). By deletion resilience (Observation 4), $E[k(Y_t)] \geq E[k(X_t)]$. Now by Lemmas 10 and 11, $E[k(X_t)] \geq E[f(X_t)] - B \geq 0.003\,t - B$. Thus, the expected number of successes in the first t steps is at most $(\lambda'' - 0.003)t + B$, which is less than $\lambda'' t$ if t is sufficiently large. (If X_0 is the empty state, then we do not require t to be sufficiently large, because $E[f(X_t)] \geq 0.003t + B$.) □

4.1. Improvements. We choose $\lambda = 0.42$ in order to make the proof of Lemma 11 (see the appendix) as simple as possible. The lemma seems to be true for λ down to about 0.41 and presumably the parameters A and B could be tweaked to get λ slightly smaller.

5. Acknowledgment-based protocols. We will prove that every acknowledgment-based protocol is transient for all $\lambda > 0.531$; see Theorem 21 for a precise statement of this claim.

An acknowledgment-based protocol can be viewed as a system which, at every step t, decides which subset of the old messages to send. The decision is a probabilistic one dependent on the histories of the messages held. As a technical device for proving our bounds, we introduce the notion of a "genie," which (in general) has more freedom in making these decisions than a protocol.

Since we consider only acknowledgment-based protocols, the behavior of each new message is independent of the other messages and of the state of the system until after its first send. This is why we ignore new messages until their first send—for Poisson arrivals this is equivalent to the convention that each message sends at its arrival time.

A *genie* is a random variable over the natural numbers, dependent on the complete history (of arrivals and sends of messages) up to time $t - 1$, which gives a natural number representing the number of messages that the genie will send at time t. Note that the number of messages that the genie sends at step t is independent of the number of newly arriving messages which send at step t. Also, the genie may send any number of messages at step t—possibly even more than the number of messages that arrived during steps $1, \ldots, t-1$. It is clear that for every acknowledgment-based protocol there is a corresponding genie. However, there are genies which do not behave like any protocol; e.g., a genie may give a cumulative total number of "sends" up to time t which exceeds the actual number of arrivals up to that time.

First, we consider the class of all genies. In Lemma 16, we show that if the arrival rate, λ, exceeds 0.567, then the backlog of messages (the difference between the cumulative number of arrivals and the cumulative number of successes) tends to infinity as time goes on. This implies that no genie has capacity greater than 0.567.

To get a better result, we consider a constrained class of genies called bucket genies. An ordinary genie (as defined previously) has no control over new inputs making their first send but has complete control over any other messages. (In particular, it can even send a message if none has arrived.) A bucket genie has no control over new inputs or over the "bucket" of messages that have already tried exactly once but has complete control over any other messages. We consider a particular type of bucket genie called an "eager" bucket genie. In Lemma 18 we show that for $\lambda \geq 0.531$, the backlog tends to infinity for eager bucket genies. In Lemma 19 we show how any bucket genie (including the acknowledgment-based protocol under consideration) can be coupled with an eager bucket genie in such a way that the the arbitrary bucket genie doesn't have many more successes than the eager bucket genie. This, combined with Lemma 16 (which shows that the eager genie doesn't have enough successes), proves the theorem.

Let $I(t), G(t)$ be the number of arrivals and the genie's send value, respectively, at step t. It is convenient to introduce some indicator variables to express various outcomes at the step under consideration. We use i_0, i_1 for the events of no new arrival, or exactly one arrival, respectively, and g_0, g_1 for the events of no send and exactly one send from the genie. The indicator random variable $S(t)$ for a success at time t is given by $S(t) = i_0 g_1 + i_1 g_0$. Let $\mathrm{In}(t) = \sum_{j \leq t} I(j)$ and $\mathrm{Out}(t) = \sum_{j \leq t} S(j)$. Define $\mathrm{Backlog}(t) = \mathrm{In}(t) - \mathrm{Out}(t)$. Let $\lambda = \lambda_0 \approx 0.567$ be the (unique) root of $\lambda = e^{-\lambda}$.

LEMMA 16. *For any genie and input rate $\lambda > \lambda_0$, there exists $\varepsilon > 0$ such that*

$$\mathrm{Prob}[\mathrm{Backlog}(t) > \varepsilon t \text{ for all } t \geq T] \to 1 \text{ as } T \to \infty.$$

Proof. Let $3\varepsilon = \lambda - e^{-\lambda} > 0$. At any step t, $S(t)$ is a Bernoulli variable with expectation $0, e^{-\lambda}, \lambda e^{-\lambda}$, according to whether $G(t) > 1$, $G(t) = 1$, $G(t) = 0$, respectively, which is dominated by the Bernoulli variable with expectation $e^{-\lambda}$. Therefore $\mathrm{E}[\mathrm{Out}(t)] \leq e^{-\lambda} t$, and also $\mathrm{Prob}[\mathrm{Out}(t) - e^{-\lambda} t < \varepsilon t \text{ for all } t \geq T] \to 1$ as $T \to \infty$. (To see this note that, by a Chernoff bound, $\mathrm{Prob}[\mathrm{Out}(t) - e^{-\lambda} t \geq \varepsilon t] \leq e^{-\delta t}$ for a positive constant δ. Thus,

$$\mathrm{Prob}[\exists t \geq T \text{ such that } \mathrm{Out}(t) - e^{-\lambda} t \geq \varepsilon t] \leq \sum_{t \geq T} e^{-\delta t},$$

which goes to 0 as T goes to ∞.)

We also have $\mathrm{E}[\mathrm{In}(t)] = \lambda t$ and $\mathrm{Prob}[\lambda t - \mathrm{In}(t) \leq \varepsilon t \text{ for all } t \geq T] \to 1$ as $T \to \infty$, since $\mathrm{In}(t) = \mathrm{Poisson}(\lambda t)$.

Since

$$\begin{aligned} \mathrm{Backlog}(t) &= \mathrm{In}(t) - \mathrm{Out}(t) \\ &= (\lambda - e^{-\lambda})t + (\mathrm{In}(t) - \lambda t) + (e^{-\lambda} t - \mathrm{Out}(t)) \\ &= \varepsilon t + (\varepsilon t + \mathrm{In}(t) - \lambda t) + (\varepsilon t + e^{-\lambda} t - \mathrm{Out}(t)), \end{aligned}$$

the result follows. □

COROLLARY 17. *No acknowledgment-based protocol is recurrent for $\lambda > \lambda_0$ or has capacity greater than λ_0.*

To strengthen the above result we introduce a restricted class of genies. We think of the messages which have failed exactly once as being contained in *the bucket*. (More generally, we could consider an array of buckets, where the jth bucket contains those messages which have failed exactly j times.) A 1-*bucket genie*, here called simply

a *bucket genie*, is a genie which simulates a given protocol for the messages in the bucket and is required to choose a send value which is at least as great as the number of sends from the bucket. Thus, on a given step, some number, say b, of the messages in the bucket will decide to send. Each of these decisions is made independently by each message, which is simulating the protocol. Then the genie will choose a number $x \geq b$, which is the number of sends that it will make. As before, g_0 is the indicator for $x = 0$ and g_1 is the indicator for $x = 1$. The indicator for success is $S(t) = i_0 g_1 + i_1 g_0$. At the end of the step, the b messages from the bucket which have sent leave the bucket. Also, any new arrivals which have collided join the bucket. Note that if the messages in the bucket decide not to send (i.e., $b = 0$) and there are no new arrivals (i.e., $i_0 = 1$), then $S(t)$ can be either 1 or 0, depending on whether or not $x = 1$. No matter what x is, no messages enter or leave the bucket during this step. For such constrained genies, we can improve the bound of Corollary 17.

For the range of arrival rates we consider, an excellent strategy for a genie is to ensure that at least one message is sent at each step. Of course a bucket genie has to respect the bucket messages and is obliged sometimes to send more than one message (inevitably failing). An *eager* genie always sends at least one message, but otherwise sends as few as possible. In particular, it sends $x = \min(1, b)$.

An eager bucket genie is easy to analyze, since every arrival is blocked by the genie and enters the bucket.

Let $\lambda = \lambda_1 \approx 0.531$ be the (unique) root of $\lambda = (1 + \lambda)e^{-2\lambda}$.

LEMMA 18. *For any eager bucket genie and input rate $\lambda > \lambda_1$, there exists $\varepsilon > 0$ such that*

$$\text{Prob}[\text{Backlog}(t) > \varepsilon t \text{ for all } t \geq T] \to 1 \text{ as } T \to \infty.$$

Proof. Let *Eager* be an eager bucket genie. Let r_i be the probability that a message in the bucket sends for the first time (and hence exits from the bucket) i steps after its arrival. Assume $\sum_{i=1}^{\infty} r_i = 1$; otherwise there is a positive probability that the message never exits from the bucket, and the result follows trivially.

The generating function for the Poisson distribution with rate λ is $e^{\lambda(z-1)}$ (i.e., the coefficient of z^k in this function gives the probability of exactly k arrivals; see, e.g., [11]). Consider the sends from the bucket at step t. Since *Eager* always blocks arriving messages, the generating function for messages entering the bucket i time steps in the past, $1 \leq i \leq t$, is $e^{\lambda(z-1)}$. Some of these messages may send at step t; the generating function for the number of sends is $e^{\lambda[(1-r_i)+r_i z-1]} = e^{\lambda r_i(z-1)}$. Finally, the generating function for all sends from the bucket at step t is the convolution of all these functions, i.e.,

$$\prod_{i=1}^{t} \exp(\lambda r_i(z-1)) = \exp\left[\lambda(z-1)\sum_{i=1}^{t} r_i\right].$$

For any $\delta > 0$, we can choose t sufficiently large so that $\sum_{i=1}^{t} r_i > 1 - \delta$. The number of sends from the bucket at step t is distributed as Poisson(λ'), where $(1-\delta)\lambda < \lambda' \leq \lambda$. The number of new arrivals sending at step t is independently Poisson(λ). The only situation in which a message succeeds under *Eager* is when there are no new arrivals and the number of sends from the bucket is zero or one. Thus the success probability at step t is $e^{-\lambda}e^{-\lambda'}(1 + \lambda')$. For sufficiently small δ, we have $\lambda_1 < \lambda' \leq \lambda$, and so $e^{-\lambda'}(1 + \lambda') < e^{-\lambda_1}(1 + \lambda_1) = e^{\lambda_1}\lambda_1 < e^{\lambda}\lambda$. Hence $e^{-\lambda}e^{-\lambda'}(1 + \lambda') \leq \lambda - 3\epsilon$ for ϵ sufficiently small. Thus the success event is dominated

by a Bernoulli variable with expectation $\lambda - 3\epsilon$. Hence, as in the previous lemma,

$$\text{Prob}[\text{Backlog}(t) > \varepsilon t \text{ for all } t \geq T] \to 1 \text{ as } T \to \infty,$$

completing the proof. □

Let *Any* be an arbitrary bucket genie and let *Eager* be the eager bucket genie based on the same bucket parameters. We may couple the executions of *Eager* and *Any* so that the same arrival sequences are presented to each. At any stage the set of messages in *Any*'s bucket is a subset of those in *Eager*'s bucket, with any differences arising from steps when there is exactly one arrival, there are no sends from the bucket, and *Eager* sends but *Any* is silent. We may further couple the behavior of the common subset of messages.

Let $\lambda = \lambda_2 \approx 0.659$ be the (unique) root of $\lambda = 1 - \lambda e^{-\lambda}$.

LEMMA 19. *For the coupled genies* Any *and* Eager *defined above, if* Out_A *and* Out_E *are the corresponding output functions, we define*

$$\Delta\text{Out}(t) = \text{Out}_E(t) - \text{Out}_A(t).$$

For any $\lambda \leq \lambda_2$ *and any* $\varepsilon > 0$,

$$\text{Prob}[\Delta\text{Out}(t) \geq -\varepsilon t \text{ for all } t \geq T] \to 1 \text{ as } T \to \infty.$$

Proof. Let c_0 be the indicator for the event that no common messages are sent. Let c_1 be the indicator for the event that exactly one common message is sent. Let c_* be the indicator for the event that more than one common message is sent. In addition, for the messages which are only in *Eager*'s bucket, we use the similar indicators e_0, e_1, e_*. Let a_0, a_1 represent *Any* not sending, or sending, *additional* messages, respectively. (Note that *Eager*'s behavior is fully determined since it will always send exactly one additional message if none of the messages in its bucket send. Otherwise, it will send no additional messages.)

We write $Z(t)$ for $\Delta\text{Out}(t) - \Delta\text{Out}(t-1)$, for $t > 0$, so Z represents the difference in success between *Eager* and *Any* in one step. In terms of the indicators we have

$$\begin{aligned} Z(t) &= S_E(t) - S_A(t) \\ &= i_0 g_{E1}(t) + i_1 g_{E0}(t) - i_0 g_{A1}(t) - i_1 g_{A0}(t), \end{aligned}$$

where $S_E(t)$ is the indicator random variable for a success of *Eager* at time t and $g_{E1}(t)$ is the event that *Eager* sends exactly one message during step t (and so on) as in the paragraph before Lemma 16. Thus,

$$Z(t) \geq i_0 c_0 (a_0(e_0 + e_1) - a_1 e_*) - i_0 c_1 (e_1 + e_*) - i_1 c_0 a_0.$$

Note that if the number of arrivals plus the number of common bucket sends is more than 1, then neither genie can succeed. We also need to keep track of the number, ΔB, of extra messages in *Eager*'s bucket. At any step, at most one new extra message can arrive; the indicator for this event is $i_1 c_0 a_0$, i.e., there is a single arrival and no sends from the common bucket, so if *Any* does not send, then this message succeeds but *Eager*'s send will cause a failure. The number of "extra" messages leaving *Eager*'s bucket at any step is unbounded, given by a random variable we could show as $\mathbf{e} = 1 \cdot e_1 + 2 \cdot e_2 + \cdots$. However, $\mathbf{e}$ dominates $e_1 + e_*$ and it is sufficient to use the latter. The change at one step in the number of extra messages satisfies

$$\Delta B(t) - \Delta B(t-1) = i_1 c_0 a_0 - \mathbf{e} \leq i_1 c_0 a_0 - (e_1 + e_*).$$

Next we define $Y(t) = Z(t) - \alpha(\Delta B(t) - \Delta B(t-1))$ for some positive constant α to be chosen below. Note that $X(t) = \sum_{j=1}^{t} Y(j) = \Delta\mathrm{Out}(t) - \alpha\Delta B(t)$. We also define

$$Y'(t) = i_0c_0(a_0(e_0+e_1) - a_1e_*) - i_0c_1(e_1+e_*) - i_1c_0a_0 - \alpha(i_1c_0a_0 - (e_1+e_*))$$

and $X'(t) = \sum_{j=1}^{t} Y'(j)$. Note that $Y(t) \geq Y'(t)$. That is, $Y(t)$ dominates $Y'(t)$.

We can identify five (exhaustive) cases A, B, C, D, E, depending on the values of the c's, a's, and e's, such that in each case $Y'(t)$ dominates a given random variable depending only on $I(t)$:

A.	c_*:	$Y'(t) \geq 0$.
B.	$(c_1 + c_0a_1)(e_1 + e_*)$:	$Y'(t) \geq \alpha - i_0$.
C.	$(c_1 + c_0a_1)e_0$:	$Y'(t) \geq 0$.
D.	$c_0a_0(e_0 + e_1)$:	$Y'(t) \geq i_0 - (1+\alpha)i_1$.
E.	$c_0a_0e_*$:	$Y'(t) \geq \alpha - (1+\alpha)i_1$.

For example, the correct interpretation of case B is "conditioned on $(c_1 + c_0a_1)(e_1 + e_*) = 1$, the value of $Y'(t)$ is at least $\alpha - i_0$." Since $\mathrm{E}[i_0] = e^{-\lambda}$ and $\mathrm{E}[i_1] = \lambda e^{-\lambda}$, we have $\mathrm{E}[Y'(t)] \geq 0$ in each case, provided that $\max\{e^{-\lambda}, \lambda e^{-\lambda}/(1-\lambda e^{-\lambda})\} \leq \alpha \leq 1/\lambda - 1$. There exists such an α for any $\lambda \leq \lambda_2$; for such λ we may take the value, say, $\alpha = e^{-\lambda}$.

Let $\mathcal{F}_t$ be the σ-field generated by the first t steps of the coupled process. Let $\hat{Y}(t) = Y'(t) - E[Y'(t) \mid \mathcal{F}_{t-1}]$ and let $\hat{X}(t) = \sum_{i=1}^{t} \hat{Y}(t)$. The sequence $\hat{X}(0), \hat{X}(1), \ldots$ forms a *martingale* (see Definition 4.11 of [21]) since $E[\hat{X}(t) \mid \mathcal{F}_{t-1}] = \hat{X}(t-1)$. Furthermore, there is a positive constant c such that $|\hat{X}(t) - \hat{X}(t-1)| \leq c$. Thus, we can apply the Hoeffding–Azuma inequality (see Theorem 4.16 of [21]).

THEOREM 20 (Hoeffding–Azuma). *Let $X_0, X_1, \ldots$ be a martingale sequence such that for each k*

$$|X_k - X_{k-1}| \leq c_k,$$

where c_k may depend upon k. Then, for all $t \geq 0$ and any $\lambda > 0$,

$$\mathrm{Prob}[\,|X_t - X_0| \geq \lambda] \leq 2\exp\left(-\frac{\lambda^2}{2\sum_{k=1}^{t} c_k^2}\right).$$

In particular, we can conclude that

$$\mathrm{Prob}[\hat{X}_t \leq -\epsilon t] \leq 2\exp\left(-\frac{\epsilon^2 t}{2c^2}\right).$$

Our choice of α above ensured that $E[Y'(t) \mid \mathcal{F}_{t-1}] \geq 0$. Hence $Y'(t) \geq \hat{Y}(t)$ and $X'(t) \geq \hat{X}(t)$. We observed earlier that $X(t) \geq X'(t)$. Thus, $X(t) \geq \hat{X}(t)$ so we have

$$\mathrm{Prob}[X_t \leq -\epsilon t] \leq 2\exp\left(-\frac{\epsilon^2 t}{2c^2}\right).$$

Since $\sum_{t\geq 0} 2\exp(-\frac{\epsilon^2 t}{2c^2})$ converges, we deduce that

$$\mathrm{Prob}[X(t) \geq -\varepsilon t \text{ for all } t \geq T] \to 1 \text{ as } T \to \infty.$$

Since $\Delta\mathrm{Out}(t) = X(t) + \alpha\Delta B(t) \geq X(t)$ for all t, we obtain the required conclusion. ∎

Finally, we can prove the main results of this section.

THEOREM 21. *Let P be an acknowledgment-based protocol. Let $\lambda = \lambda_1 \approx 0.531$ be the (unique) root of $\lambda = (1+\lambda)e^{-2\lambda}$. Then*

1. *P is transient for arrival rates greater than λ_1;*
2. *P has capacity no greater than λ_1.*

Proof. Let λ be the arrival rate, and suppose $\lambda > \lambda_1$. If $\lambda > \lambda_0 \approx 0.567$, then the result follows from Lemma 16. Otherwise, we can assume that $\lambda < \lambda_2 \approx 0.659$. If E is the eager genie derived from P, then the corresponding Backlogs satisfy $\text{Backlog}_P(t) = \text{Backlog}_E(t) + \Delta\text{Out}(t)$. The results of Lemmas 18 and 19 show that, for some $\varepsilon > 0$, both $\text{Prob}[\text{Backlog}_E(t) > 2\varepsilon t \text{ for all } t \geq T]$ and $\text{Prob}[\Delta\text{Out}(t) \geq -\varepsilon t \text{ for all } t \geq T]$ tend to 1 as $T \to \infty$. The conclusion of the theorem follows. □

Appendix. Proof of Lemma 11. Let $j(r,p) = -h(r,p)$. We will show that for any $r \in [0,1]$ and $p \in [0,1]$, $j(r,p) \leq -0.003$.

Case 1. *$r + p - rp \geq r \geq B/A$ and $p \geq r$.* In this case we have

$$g(r,p) = e^{-\lambda}((1-r)p + (1-p)r + (1-p)(1-r)\lambda),$$
$$j(r,p) = g(r,p) - \lambda.$$

Observe that

$$j(r,p) = e^{-\lambda} \sum_{i=0}^{1} \sum_{j=0}^{1} c_{i,j} p^i r^j,$$

where the coefficients $c_{i,j}$ are defined as follows:

$$c_{0,0} = \lambda(1 - e^{\lambda}),$$
$$c_{1,0} = 1 - \lambda,$$
$$c_{0,1} = 1 - \lambda,$$
$$c_{1,1} = -2 + \lambda.$$

Note that the only positive coefficients are $c_{1,0}$ and $c_{0,1}$. Thus, if $p \in [p_1, p_2]$ and $r \in [r_1, r_2]$, then $j(r,p)$ is at most $\mathcal{U}(p_1, p_2, r_1, r_2)$, which we define as

$$e^{-\lambda}(c_{0,0} + c_{1,0}p_2 + c_{0,1}r_2 + c_{1,1}p_1 r_1).$$

Now we need only check that for all $r_1 \in (B/A - 0.01, 1)$ and $p_1 \in [r_1, 1)$ such that p_1 and r_1 are multiples of 0.01, $\mathcal{U}(p_1, p_1 + 0.01, r_1, r_1 + 0.01)$ is at most -0.003. This is the case. (The highest value is $\mathcal{U}(0.45, 0.46, 0.45, 0.46)$, which is -0.00366228.)

Case 2. *$r + p - rp \geq r \geq B/A$ and $p < r$.* Now we have

$$g(r,p) = e^{-\lambda}\left((1-r)p + (1-p)\frac{r - r^2/2}{1 - p/2} + (1-p)(1-r)\lambda\right),$$
$$j(r,p) = g(r,p) - \lambda.$$

Observe that

$$(1 - p/2)j(r,p) = e^{-\lambda} \sum_{i=0}^{2} \sum_{j=0}^{2} c_{i,j} p^i r^j,$$

where the coefficients $c_{i,j}$ are defined as follows:

$$\begin{aligned}
c_{0,0} &= \lambda(1-e^{\lambda}),\\
c_{1,0} &= 1-3\lambda/2+e^{\lambda}\lambda/2,\\
c_{0,1} &= 1-\lambda,\\
c_{1,1} &= -2+3\lambda/2,\\
c_{2,0} &= -1/2+\lambda/2,\\
c_{0,2} &= -1/2,\\
c_{2,1} &= 1/2-\lambda/2,\\
c_{1,2} &= 1/2,\\
c_{2,2} &= 0.
\end{aligned}$$

Note that the only positive coefficients are $c_{1,0}$, $c_{0,1}$, $c_{2,1}$, and $c_{1,2}$. Thus, if $p \in [p_1, p_2]$ and $r \in [r_1, r_2]$, then $j(r,p)$ is at most $\mathcal{U}(p_1, p_2, r_1, r_2)$, which we define as

$$\frac{c_{0,0}+c_{1,0}p_2+c_{0,1}r_2+c_{1,1}p_1r_1+c_{2,0}p_1^2+c_{0,2}r_1^2+c_{2,1}p_2^2r_2+c_{1,2}p_2r_2^2+c_{2,2}p_1^2r_1^2}{e^{\lambda}(1-p_2/2)}.$$

Now we need only check that for all $r_1 \in (B/A-0.005, 1)$ and $p_1 \in [0, r_1]$ such that p_1 and r_1 are multiples of 0.005, $\mathcal{U}(p_1, p_1+0.005, r_1, r_1+0.005)$ is at most -0.003. This is the case. (The highest value for these parameters is $\mathcal{U}(0.45, 0.455, 0.455, 0.46) = -0.00479648$.)

Case 3. $r+p-rp \geq B/A \geq r$ *and* $p \geq r$. In this case we have

$$\begin{aligned}
g(r,p) &= e^{-\lambda}((1-r)p+(1-p)r+(1-p)(1-r)\lambda),\\
j(r,p) &= g(r,p)-\lambda-(-Ar+B)e^{-\lambda}(1-r)p.
\end{aligned}$$

Observe that

$$j(r,p) = e^{-\lambda}\sum_{i=0}^{2}\sum_{j=0}^{2} c_{i,j}p^i r^j,$$

where the coefficients $c_{i,j}$ are defined as follows:

$$\begin{aligned}
c_{0,0} &= \lambda(1-e^{\lambda}),\\
c_{1,0} &= 1-B-\lambda,\\
c_{0,1} &= 1-\lambda,\\
c_{1,1} &= -2+A+B+\lambda,\\
c_{2,0} &= 0,\\
c_{0,2} &= 0,\\
c_{2,1} &= 0,\\
c_{1,2} &= -A,\\
c_{2,2} &= 0.
\end{aligned}$$

Note that the only positive coefficients are $c_{1,0}$ and $c_{0,1}$. Thus, if $p \in [p_1, p_2]$ and $r \in [r_1, r_2]$, then $j(r,p)$ is at most $\mathcal{U}(p_1, p_2, r_1, r_2)$, which we define as

$$\frac{c_{0,0}+c_{1,0}p_2+c_{0,1}r_2+c_{1,1}p_1r_1+c_{2,0}p_1^2+c_{0,2}r_1^2+c_{2,1}p_1^2r_1+c_{1,2}p_1r_1^2+c_{2,2}p_1^2r_1^2}{e^{\lambda}}.$$

Now we need only check that for all $p_1 \in [0,1)$ and $r_1 \in [0,p_1]$ such that p_1 and r_1 are multiples of 0.01, $\mathcal{U}(p_1, p_1+0.01, r_1, r_1+0.01)$ is at most -0.003. This is the case. (The highest value is $\mathcal{U}(0.44, 0.45, 0.44, 0.45) = -0.00700507$.)

Case 4. $r + p - rp \geq B/A \geq r$ *and* $p < r$. Now we have

$$g(r,p) = e^{-\lambda}\left((1-r)p + (1-p)\frac{r - r^2/2}{1-p/2} + (1-p)(1-r)\lambda\right),$$
$$j(r,p) = g(r,p) - \lambda - (-Ar + B)e^{-\lambda}(1-r)p.$$

Observe that

$$j(r,p) = e^{-\lambda}(1/2)\frac{\sum_{i=0}^{2}\sum_{j=0}^{2} c_{i,j}p^i r^j}{1-p/2},$$

where the coefficients $c_{i,j}$ are defined as follows:

$$\begin{aligned}
c_{0,0} &= 2\lambda(1-e^{\lambda}),\\
c_{1,0} &= 2 - 2B - 3\lambda + \lambda e^{\lambda},\\
c_{0,1} &= 2 - 2\lambda,\\
c_{1,1} &= -4 + 2A + 2B + 3\lambda,\\
c_{2,0} &= -1 + B + \lambda,\\
c_{0,2} &= -1,\\
c_{2,1} &= 1 - A - B - \lambda,\\
c_{1,2} &= 1 - 2A,\\
c_{2,2} &= A.
\end{aligned}$$

Note that the coefficients are all negative except $c_{1,0}$, $c_{0,1}$, and $c_{2,2}$. Thus, if $p \in [p_1, p_2]$ and $r \in [r_1, r_2]$, then $j(r,p)$ is at most $\mathcal{U}(p_1, p_2, r_1, r_2)$, which we define as

$$\frac{c_{0,0} + c_{1,0}p_2 + c_{0,1}r_2 + c_{1,1}p_1r_1 + c_{2,0}p_1^2 + c_{0,2}r_1^2 + c_{2,1}p_1^2r_1 + c_{1,2}p_1r_1^2 + c_{2,2}p_2^2r_2^2}{2e^{\lambda}(1-p_2/2)}.$$

Now we need only check that for all $p_1 \in [0,1)$ and $r_1 \in [p_1,1)$ such that p_1 and r_1 are multiples of 0.01, $\mathcal{U}(p_1, p_1+0.01, r_1, r_1+0.01)$ is at most -0.003. This is the case. (The highest value is $\mathcal{U}(0.44, 0.45, 0.44, 0.45) = -0.00337716$.)

Case 5. $B/A \geq r + p - rp \geq r$ *and* $p \geq r$. In this case we have

$$\begin{aligned}
g(r,p) &= e^{-\lambda}((1-r)p + (1-p)r + (1-p)(1-r)\lambda),\\
j(r,p) &= g(r,p) - \lambda\\
&\quad + (-A(r+p-rp) + B)(1 - (1-r)(1-p)e^{-\lambda}(1+\lambda))\\
&\quad - (-Ar + B)e^{-\lambda}(1-r)p.
\end{aligned}$$

Observe that

$$j(r,p) = e^{-\lambda}\sum_{i=0}^{2}\sum_{j=0}^{2} c_{i,j}p^i r^j,$$

where the coefficients $c_{i,j}$ are defined as follows:

$$\begin{aligned}
c_{0,0} &= -B + Be^{\lambda} + \lambda - B\lambda - e^{\lambda}\lambda,\\
c_{1,0} &= 1 + A - Ae^{\lambda} - \lambda + A\lambda + B\lambda,\\
c_{0,1} &= 1 + A + B - Ae^{\lambda} - \lambda + A\lambda + B\lambda,\\
c_{1,1} &= -2 - 2A + Ae^{\lambda} + \lambda - 3A\lambda - B\lambda,\\
c_{2,0} &= -A - A\lambda,\\
c_{0,2} &= -A - A\lambda,\\
c_{2,1} &= 2A + 2A\lambda,\\
c_{1,2} &= A + 2A\lambda,\\
c_{2,2} &= -A - A\lambda.
\end{aligned}$$

Note that the only positive coefficients are $c_{1,0}$, $c_{0,1}$, $c_{2,1}$, and $c_{1,2}$. Thus, if $p \in [p_1, p_2]$ and $r \in [r_1, r_2]$, then $j(r,p)$ is at most $\mathcal{U}(p_1, p_2, r_1, r_2)$, which we define as

$$\frac{c_{0,0} + c_{1,0}p_2 + c_{0,1}r_2 + c_{1,1}p_1r_1 + c_{2,0}p_1^2 + c_{0,2}r_1^2 + c_{2,1}p_2^2r_2 + c_{1,2}p_2r_2^2 + c_{2,2}p_1^2r_1^2}{e^{\lambda}}.$$

Now we need only check that for all $p_1 \in [0,1)$ and $r_1 \in [0, p_1]$ such that p_1 and r_1 are multiples of 0.01, $\mathcal{U}(p_1, p_1 + 0.01, r_1, r_1 + 0.01)$ is at most -0.003. This is the case. (The highest value is $\mathcal{U}(0.19, 0.2, 0.19, 0.2) = -0.0073656$.)

Case 6. $B/A \geq r + p - rp \geq r$ *and* $p < r$. Now we have

$$\begin{aligned}
g(r,p) &= e^{-\lambda}\left((1-r)p + (1-p)\frac{r - r^2/2}{1 - p/2} + (1-p)(1-r)\lambda\right),\\
j(r,p) &= g(r,p) - \lambda\\
&\quad + (-A(r + p - rp) + B)(1 - (1-r)(1-p)e^{-\lambda}(1+\lambda))\\
&\quad - (-Ar + B)e^{-\lambda}(1-r)p.
\end{aligned}$$

Observe that

$$(1 - p/2)j(r,p) = e^{-\lambda}\sum_{i=0}^{3}\sum_{j=0}^{2} c_{i,j}p^i r^j,$$

where the coefficients $c_{i,j}$ are defined as follows:

$$\begin{aligned}
c_{0,0} &= -B + Be^{\lambda} + \lambda - B\lambda - e^{\lambda}\lambda,\\
c_{1,0} &= 1 + A + B/2 - Ae^{\lambda} - Be^{\lambda}/2 - 3\lambda/2 + A\lambda + 3B\lambda/2 + e^{\lambda}\lambda/2,\\
c_{0,1} &= 1 + A + B - Ae^{\lambda} - \lambda + A\lambda + B\lambda,\\
c_{1,1} &= -2 - 5A/2 - B/2 + 3Ae^{\lambda}/2 + 3\lambda/2 - 7A\lambda/2 - 3B\lambda/2,\\
c_{2,0} &= -1/2 - 3A/2 + Ae^{\lambda}/2 + \lambda/2 - 3A\lambda/2 - B\lambda/2,\\
c_{0,2} &= -1/2 - A - A\lambda,\\
c_{2,1} &= 1/2 + 3A - Ae^{\lambda}/2 - \lambda/2 + 7A\lambda/2 + B\lambda/2,\\
c_{1,2} &= 1/2 + 3A/2 + 5A\lambda/2,\\
c_{2,2} &= -3A/2 - 2A\lambda,\\
c_{3,0} &= A/2 + A\lambda/2,\\
c_{3,1} &= -A - A\lambda,\\
c_{3,2} &= A/2 + A\lambda/2.
\end{aligned}$$

Note that the only positive coefficients are $c_{1,0}$, $c_{0,1}$, $c_{2,1}$, $c_{1,2}$, $c_{3,0}$, and $c_{3,2}$. Thus, if $p \in [p_1, p_2]$ and $r \in [r_1, r_2]$, then $j(r,p)$ is at most $\mathcal{U}(p_1, p_2, r_1, r_2)$, which we define as

$$c_{0,0} + c_{1,0}p_2 + c_{0,1}r_2 + c_{1,1}p_1r_1 + c_{2,0}p_1^2 + c_{0,2}r_1^2 + c_{2,1}p_2^2r_2 + c_{1,2}p_2r_2^2 + c_{2,2}p_1^2r_1^2 + c_{3,0}p_2^3 + c_{3,1}p_1^3r_1 + c_{3,2}p_2^3r_2^2$$

divided by $e^\lambda(1 - p_2/2)$.

Now we need only check that for all $p_1 \in [0, 1)$ and $r_1 \in [p_1, 1)$ such that p_1 and r_1 are multiples of 0.005, $\mathcal{U}(p_1, p_1 + 0.005, r_1, r_1 + 0.005)$ is at most -0.003. This is the case. (The highest value is $\mathcal{U}(0.01, 0.015, 0.3, 0.305) = -0.00383814$.)

REFERENCES

[1] D. Aldous, *Ultimate instability of exponential back-off protocol for acknowledgement-based transmission control of random access communication channels*, IEEE Trans. Inform. Theory, 33 (1987), pp. 219–233.

[2] H. Al-Ammal, L.A. Goldberg, and P. MacKenzie, *An improved stability bound for binary exponential backoff*, Theory Comput. Syst., 34 (2001), pp. 229–244.

[3] J.I. Capetanakis, *Tree algorithms for packet broadcast channels*, IEEE Trans. Inform. Theory, 25 (1979), pp. 505–515.

[4] A. Ephremides and B. Hajek, *Information theory and communication networks: An unconsummated union*, IEEE Trans. Inform. Theory, 44 (1998), pp. 2416–2432.

[5] G. Fayolle, V.A. Malyshev, and M.V. Menshikov, *Topics in the Constructive Theory of Countable Markov Chains*, Cambridge University Press, Cambridge, UK, 1995.

[6] L.A. Goldberg and P.D. MacKenzie, *Analysis of practical backoff protocols for contention resolution with multiple servers*, J. Comput. System Sci., 58 (1999), pp. 232–258.

[7] F.G. Foster, *On the stochastic matrices associated with certain queueing processes*, Ann. Math. Statist., 24 (1953), pp. 355–360.

[8] L.A. Goldberg, P.D. MacKenzie, M. Paterson, and A. Srinivasan, *Contention resolution with constant expected delay*, J. ACM, 47 (2000), pp. 1048–1096.

[9] J. Goodman, A.G. Greenberg, N. Madras, and P. March, *Stability of binary exponential backoff*, J. ACM, 35 (1988), pp. 579–602.

[10] A.G. Greenberg, P. Flajolet and R. Ladner, *Estimating the multiplicities of conflicts to speed their resolution in multiple access channels*, J. ACM, 34 (1987), pp. 289–325.

[11] G.R. Grimmett and D.R. Stirzaker, *Probability and Random Processes*, 2nd ed., Oxford University Press, New York, 1992.

[12] J. Håstad, T. Leighton, and B. Rogoff, *Analysis of backoff protocols for multiple access channels*, SIAM J. Comput., 25 (1996), pp. 740–774.

[13] F.P. Kelly, *Stochastic models of computer communication systems*, J. Roy. Statist. Soc. Ser. B, 47 (1985), pp. 379–395.

[14] F.P. Kelly and I.M. MacPhee, *The number of packets transmitted by collision detect random access schemes*, Ann. Probab., 15 (1987), pp. 1557–1568.

[15] U. Loher, *Efficiency of first-come first-served algorithms*, in Proceedings of the 1998 IEEE International Symposium on Information Theory, IEEE Information Theory Society, 1998, p. 108.

[16] U. Loher, *Information-Theoretic and Genie-Aided Analyses of Random-Access Algorithms*, Ph.D. Thesis, DISS ETH 12627, Swiss Federal Institute of Technology, Zurich, 1998.

[17] I.M. MacPhee, *On Optimal Strategies in Stochastic Decision Processes*, D. Phil. Thesis, University of Cambridge, Cambridge, UK, 1987.

[18] R.M. Metcalfe and D.R. Boggs, *Ethernet: Distributed packet switching for local computer networks*, Comm. ACM, 19 (1976), pp. 395–404.

[19] M. Molle and G.C. Polyzos, *Conflict Resolution Algorithms and Their Performance Analysis*, Technical Report CS93-300, Computer Systems Research Institute, University of Toronto, 1993.

[20] J. Mosely and P.A. Humblet, *A class of efficient contention resolution algorithms for multiple access channels*, IEEE Trans. Commun., 33 (1985), pp. 145–151.

[21] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[22] P. Raghavan and E. Upfal, *Stochastic contention resolution with short delays*, SIAM J. Comput., 28 (1998), pp. 709–719.

[23] R. Rom and M. Sidi, *Multiple Access Protocols: Performance and Analysis*, Springer-Verlag, New York, 1990.

[24] B.S. Tsybakov and N.B. Likhanov, *An upper bound for the capacity of a random multiple-access system*, Problemy Peredachi Informatsii, 23 (1987), pp. 64–78.

[25] B.S. Tsybakov and V.A. Mikhailov, *Free synchronous packet access in a broadcast channel with feedback*, Problems Inform. Transmission, 14 (1978), pp. 259–280.

[26] S. Verdu, *Computation of the efficiency of the Mosely-Humblet contention resolution algorithm: A simple method*, Proc. IEEE, 74 (1986), pp. 613–614.

[27] N.S. Vvedenskaya and M.S. Pinsker, *Non-optimality of the part-and-try algorithm*, in Abstracts of the International Workshop on Convolutional Codes and Multiuser Communication, Sochi, USSR, 1983, pp. 141–144.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 332–350

STACKELBERG SCHEDULING STRATEGIES*

TIM ROUGHGARDEN†

Abstract. We study the problem of optimizing the performance of a system shared by selfish, noncooperative users. We consider the concrete setting of scheduling small jobs on a set of shared machines possessing latency functions that specify the amount of time needed to complete a job, given the machine load. We measure system performance by the *total latency* of the system.

Assigning jobs according to the selfish interests of individual users, who wish to minimize only the latency that their own jobs experience, typically results in suboptimal system performance. However, in many systems of this type there is a mixture of "selfishly controlled" and "centrally controlled" jobs. The congestion due to centrally controlled jobs will influence the actions of selfish users, and we thus aspire to contain the degradation in system performance due to selfish behavior by scheduling the centrally controlled jobs in the best possible way.

We formulate this goal as an optimization problem via *Stackelberg games*, games in which one player acts a *leader* (here, the centralized authority interested in optimizing system performance) and the rest as *followers* (the selfish users). The problem is then to compute a strategy for the leader (a *Stackelberg strategy*) that induces the followers to react in a way that (approximately) minimizes the total latency in the system.

In this paper, we prove that it is NP-hard to compute an optimal Stackelberg strategy and present simple strategies with provably good performance guarantees. More precisely, we give a simple algorithm that computes a strategy inducing a job assignment with total latency no more than a constant times that of the optimal assignment of all of the jobs; in the absence of centrally controlled jobs and a Stackelberg strategy, no result of this type is possible. We also prove stronger performance guarantees in the special case where every machine latency function is linear in the machine load.

Key words. selfish routing, Stackelberg equilibria, scheduling

AMS subject classifications. 68Q25, 68W25, 90B35, 91A65

DOI. 10.1137/S0097539701397059

1. Introduction.

Coping with selfishness. One of the most basic problems arising in the management of a set of resources is that of optimizing system performance. A concrete example of such a problem is as follows: given a large set of small jobs to be assigned to a set of machines, each with a latency function that specifies the amount of time needed to complete a job given the machine load, find the allocation of jobs to machines minimizing the *total latency* of the system.

In many such systems, including the Internet and other large-scale communication networks, there is no central authority controlling the allocation of shared resources; instead, system users are free to act in a selfish manner [7]. This observation has led many authors (e.g., [13, 26, 30, 37, 45, 48]) to model the behavior of users in such a system by a *noncooperative game* and to study the resulting *Nash equilibria*. (See [38] for an introduction to basic game-theoretic concepts.)

Motivated by the well-known fact that Nash equilibria may be *inefficient* [16]

*Received by the editors October 29, 2001; accepted for publication (in revised form) April 17, 2003; published electronically January 22, 2004DATE. A preliminary version of this paper appeared in *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, 2001. This research was supported by an NSF Graduate Fellowship, a Cornell University Fellowship, and ONR grant N00014-98-1-0589.

http://www.siam.org/journals/sicomp/33-2/39705.html

†Department of Computer Science, Cornell University, Ithaca, NY 14853. Current address: Computer Science Division, UC Berkeley, Soda Hall, Berkeley, CA 94720 (timr@cs.berkeley.edu).

(i.e., they need not optimize system performance), researchers have proposed several different approaches for *coping with selfishness*—that is, for ensuring that selfish behavior results in a desirable outcome. Recent examples abound in the computer science literature: to name a few, Korilis, Lazar, and Orda [27, 28] give methods for improving system performance by adding additional capacity to system resources, Cocchi et al. [8] and Cole, Dodis, and Roughgarden [9, 10] investigate the control of selfish users through various pricing policies, Shenker [48] demonstrates that an appropriate (centralized) protocol at a network switch induces selfish users to exhibit good flow control behavior, and Roughgarden [42] studies the problem of designing networks that exhibit good performance when used selfishly.

A related area of research concerned with controlling selfish behavior with efficient algorithms is that of *algorithmic mechanism design* [2, 19, 35, 36, 41], which in turn is inspired by classical mechanism design (see, e.g., Mas-Colell, Whinston, and Green [33, Chap. 23]). In this setting, an algorithm is designed to collect data from users and compute an outcome using this data. For example, the algorithm might compute the set of users that will receive some good, perhaps a movie multicast over the Internet, based on the bids of the users of the system. The difficulty of these problems stems from the assumptions that the algorithm is publicly known and that users are selfish and may report false data, if doing so improves their personal objective function. This problem is typically resolved via a *payment scheme*, where the algorithm distributes payments to users according to the outcome and the data collected so that all users have a strong incentive to report truthful data.

In this paper, we pursue a different approach. In many systems, there will be a mix of "selfishly controlled" and "centrally controlled" jobs—that is, the shared resource is used by both selfish individuals and some central authority. For example, clients of a system may be charged at two different prices: clients paying the higher price are given access to the system and the ability to schedule their own tasks (presumably in a way that accomplishes them as quickly as possible), while clients paying only the "bargain rate" can use the system but have no control over which resources their jobs consume (and thus these jobs qualify as centrally controlled). A more concrete example of such a system arises in networks that allow a large customer to set up a so-called *virtual private network* consisting of guaranteed and preassigned virtual paths for ongoing use [6, 17, 20, 22, 26]. The bandwidth needed for the virtual private network may be viewed as centrally controlled (its routes may be chosen by the network manager), while individual users of the network continue to behave in a selfish and independent fashion.

We investigate the following question: Given a system with centrally and selfishly controlled jobs, *how should centrally controlled jobs be assigned to resources to induce "good" (albeit selfish) behavior from the noncooperative users?* This approach has several appealing aspects: no communication is required between the system users and an algorithm, no notion of currency is needed, no resources need to be added to or removed from the system, and the assignment of centrally controlled jobs is often easily modified as the amount of job traffic evolves over time.

Stackelberg games. We are thus led to consider a type of game in which the roles of different players are asymmetric. One player, responsible for assigning the centrally controlled jobs to resources and interested in optimizing social welfare, acts as a *leader*. The leader may hold its assignment (its *strategy*) fixed while all other agents (the *followers*) react independently and selfishly to the leader's strategy, reaching a Nash equilibrium relative to the leader's strategy. These types of games, called

Stackelberg games, and the resulting *Stackelberg equilibria* have been well studied in the game theory literature (see, e.g., [3, sect. 2.3] or [4, sect. 3.6] for an introduction and [49] for their origin) and have been previously studied in the contexts of both competitive facility location [40] and networking [14, 15, 18, 26]. With the exception of [26], however, the leader/follower hierarchy has been used to model classes of selfish agents with different priority levels; this setting differs from ours in that no agent is interested in optimizing system performance.[1] The paper of Korilis, Lazar, and Orda [26], while more similar in spirit to ours, focuses on deriving necessary and sufficient conditions (on the number of selfish users, the fraction of the job traffic that is centrally controlled, etc.) for the existence of a leader strategy inducing an optimal assignment of jobs to resources; moreover, only one type of latency function is considered. By contrast, we are interested in simple leader strategies that *always* induce optimal or near-optimal behavior from the system users for *any* set of latency functions.

Problems studied in this paper. We focus on the problem, described at the beginning of the paper, of scheduling jobs on a set of machines with load-dependent latencies in order to minimize the total latency. In addition to being one of the most commonly studied models [11, 12, 18, 26, 29, 30, 34, 35, 36, 41], occasionally in the equivalent formulation of routing on a set of parallel links, the simple setting of scheduling jobs on machines permits a study of the effects of different leader strategies in Stackelberg games without any additional complications, such as the issue of path-selection in a complicated network. We also focus on a scenario in which there is a large number of jobs, each of very small size. We note that this differs from nearly all scheduling research in the theoretical computer science and discrete optimization communities (surveyed in, for example, Hall [23]). This assumption is, however, consistent with a large body of existing literature on Nash equilibria in congested systems—see [5, 13, 44, 47] and the references therein—and greatly aids our analysis. For example, this assumption ensures the existence and essential uniqueness of the equilibrium reached by selfish users relative to any Stackelberg strategy. While it is clearly desirable to study more general networks as well as jobs of nonnegligible size, we nonetheless feel that the simple model considered in this paper is sufficient to illustrate the main issues that arise in centrally scheduling traffic in the presence of selfish users.

We may now restate our central questions quantitatively:

(1) Among all leader strategies for a given set of machines and jobs, can we characterize and/or compute the strategy inducing the *Stackelberg equilibrium*—i.e., the equilibrium of minimum total latency?

(2) What is the worst-case ratio between the total latency of the Stackelberg equilibrium and that of the optimal assignment of jobs to machines?

Our results. We give a simple polynomial-time algorithm for computing a leader strategy that induces a job assignment with total latency no more than $\frac{1}{\alpha}$ times that of the optimal assignment of jobs to machines, where α denotes the fraction of jobs that are centrally controlled. We also exhibit, for each value of α, an instance in which *no* strategy can achieve a better performance guarantee. This result stands in sharp contrast to known results about *Nash* equilibria in this model; in particular, the total latency of the Nash equilibrium may be arbitrarily larger than that of the

[1] Typically, Stackelberg games model *selfish* agents with asymmetric roles; our use of them is somewhat unconventional.

optimal assignment of jobs to machines, even in the special case of two machines [45].

In the well-studied special case where every machine possesses a latency function linear in the congestion, we give a simple $O(m^2)$ algorithm for computing a strategy inducing a job assignment with total latency no more than $\frac{4}{3+\alpha}$ times that of the optimal assignment, where α is the fraction of centrally controlled jobs, and m is the number of machines. We again give instances in which no strategy can provide a stronger guarantee.

Finally, we consider the optimization problem of computing the strategy inducing the Stackelberg equilibrium and show that it is NP-hard, even in the special case where every latency function is linear.

We note that our results give a (sharp) trade-off between the optimal assignment and the Nash equilibrium, as a function of the fraction of centrally controlled jobs, in the following sense. Roughgarden and Tardos [45], motivated by a paper of Koutsoupias and Papadimitriou [30], showed that in general the Nash equilibrium can be arbitrarily more costly than the optimal assignment, but if every machine latency function is linear, then the total latency of the Nash equilibrium is no more than $\frac{4}{3}$ times that of the optimal assignment. Thus, our results reduce to those of [45] when $\alpha = 0$, give the trivial result that the Stackelberg equilibrium for $\alpha = 1$ is the optimal assignment, and quantify the worst possible ratio between the cost of the Stackelberg equilibrium—in some sense, a "mixture" of the Nash equilibrium and the optimal assignment—and the cost of the optimal assignment for all intermediate values of α.

Our approach also adds an algorithmic dimension to the existing studies comparing Nash equilibria and optimal solutions [1, 11, 12, 24, 29, 30, 34, 43, 45, 46], in that one aspect of our analysis of Stackelberg equilibria is the design of algorithms for efficiently computing good Stackelberg strategies. Further, while Nash and optimal assignments in our model can be characterized and computed efficiently via convex programming [13, 45]—a fact that is crucial for existing comparisons of optima and equilibria [43, 45, 46]—our hardness result implies that no such characterization of Stackelberg equilibria is possible. With the central approach of earlier works ruled out, new techniques are required for our results.

Organization. In section 2 we formalize our model and state several preliminary lemmas. In section 3 we introduce three simple algorithms for computing Stackelberg strategies. In sections 4 and 5, we prove that our third algorithm achieves the best possible worst-case performance guarantee for instances with general and linear latency functions, respectively. In section 6, we prove that computing the optimal strategy is NP-hard, even when every latency function is linear. Section 7 concludes with directions for future work.

2. Preliminaries.

2.1. The model. We consider a set M of m machines $1, 2, \ldots, m$, where machine i is endowed with a *latency function* $\ell_i(\cdot)$ that measures the load-dependent time required to complete a job. We require that each latency function be nonnegative, continuous, and nondecreasing in its argument. For our algorithms to be implemented efficiently, we also require the weak condition that $x_i \cdot \ell_i(x_i)$ is a convex function for each machine i (where x_i denotes the machine load).[2] We assume a finite and positive *rate* r of job arrivals; an *assignment* of the jobs to the machines is an m-vector $x \in \mathcal{R}_+^m$

[2]Thus, $\ell_i(x_i)$ may be any convex function, or $\log(1 + x_i)$, etc. Continuous approximations of step functions, however, do not satisfy this condition.

such that $\sum_{i=1}^{m} x_i = r$. When we are interested in the total load on a subset $M' \subseteq M$ of the machines, we write $x(M') = \sum_{i \in M'} x_i$. We measure system performance via the *cost* or *total latency* $C(x)$ of an assignment x, defined by $C(x) = \sum_{i=1}^{m} x_i \ell_i(x_i)$. We note that all jobs assigned to the same machine experience the same latency; again, this differs from much of the traditional scheduling literature but agrees with common models of equilibria in congested systems where a particular allocation of resources represents a "steady-state solution" with jobs arriving continuously over time.

We will consider instances with and without centrally controlled jobs. We denote an instance with machines M, rate r, and no centrally controlled jobs by (M, r). An instance with centrally controlled jobs (a *Stackelberg instance*) will be denoted by (M, r, α), where the third parameter $\alpha \in (0, 1)$ indicates the fraction of the overall traffic that is centrally controlled.

2.2. Nash equilibria and optimal assignments. If jobs are generated and assigned to machines by noncooperative agents who wish to minimize the amount of time it takes for their work to complete, we expect the assignment to be "stable" or "at equilibrium" in the following sense: no job can strictly decrease the latency it experiences by changing machines. The following definition is motivated by this notion of a stable assignment by noncooperative agents.

DEFINITION 2.1. *An assignment x to M is* at Nash equilibrium *(or is a* Nash assignment*) if whenever $i, j \in M$ with $x_i > 0$, $\ell_i(x_i) \leq \ell_j(x_j)$.*

In particular, all machines in use by an assignment at Nash equilibrium have equal latency. We may thus express the cost of a Nash assignment in the following simple form.

LEMMA 2.2. *If x is an assignment at Nash equilibrium for (M, r) such that all machines in use have common latency L, then*

$$C(x) = rL.$$

Example 2.3. An assignment at Nash equilibrium does not in general optimize the system performance. To see this, consider a two-machine example, in which the first machine has constant latency function $\ell_1(x_1) = 1$ and the second has latency function $\ell_2(x_2) = x_2$. If we put $r = 1$, we see that the (optimal) assignment $(\frac{1}{2}, \frac{1}{2})$ has total latency $\frac{3}{4}$, whereas the (unique) assignment at Nash equilibrium assigns all work to the second machine, thereby incurring a cost of 1.

We end our preliminary discussion of Nash assignments by noting that they exist and are essentially unique.

LEMMA 2.4 (see [5, 13, 45]). *Suppose M is a set of machines with continuous, nondecreasing latency functions. Then*

(a) *for any rate $r > 0$ of job traffic, there exists an assignment of jobs to M at Nash equilibrium;*

(b) *if x, x' are assignments at Nash equilibrium for (M, r), then $\ell_i(x_i) = \ell_i(x'_i)$ for each machine i.*

In particular, Definition 2.1 and Lemma 2.4(b) imply that any two Nash assignments for an instance (M, r) have equal cost.

For our final preliminary result, we give an analogous characterization of optimal assignments that will be useful in section 5. For a machine i with differentiable latency function ℓ_i, define ℓ_i^* as the marginal cost of increasing the load of machine i—formally, by $\ell_i^*(x_i) = [\frac{d}{dy}(y \cdot \ell_i(y))](x_i) = \ell_i(x_i) + x_i \cdot \ell_i'(x_i)$. We will call ℓ_i^* the *marginal cost function* of machine i. Then the following lemma holds.

LEMMA 2.5 (see [5, 13, 45]). *Suppose M is a set of machines with differentiable latency functions ℓ and that $x_i \cdot \ell_i(x_i)$ is a convex function for each machine i. Then an assignment x to M is optimal if and only if, whenever $i, j \in M$ with $x_i > 0$, $\ell_i^*(x_i) \leq \ell_j^*(x_j)$. Moreover, the optimal assignment can be computed in polynomial time.*

Lemma 2.5 clearly gives a characterization of *locally* optimal assignments; in particular, if the condition fails, moving a few jobs from a machine with a large marginal cost to a machine with a small marginal cost yields a new assignment with smaller cost. That it also characterizes *globally* optimal solutions follows from the fact that the optimal assignment minimizes a convex function ($C(x)$) over a convex set (the polytope of assignments) and that the local and global minima of a convex function on a convex set coincide (see, for example, [39, Thm. 2.3.4]). This observation also implies that the optimal solution can be computed in polynomial time via convex programming.

Definition 2.1 and Lemma 2.5 yield the following useful corollary.

COROLLARY 2.6 (see [5, 13, 45]). *Suppose M is a set of machines with differentiable latency functions ℓ and that $x_i \cdot \ell_i(x_i)$ is a convex function for each machine i. Then an assignment x to M is optimal if and only if x is a Nash assignment with respect to latency functions ℓ^*.*

Remark 2.7. We will typically denote the optimal assignment for an instance by x^*. The marginal cost functions are denoted by ℓ^* as they are "optimal latency functions" in a sense made precise by Corollary 2.6: the optimal assignment x^* arises as an assignment at Nash equilibrium with respect to latency functions ℓ^*.

2.3. Stackelberg strategies and induced equilibria. In this subsection we define our notion of a Stackelberg game and consider two examples. Recall that we desire a hierarchical game, where a leader assigns centrally controlled jobs to machines and, holding this strategy fixed, the selfish users of the system react in a noncooperative and selfish manner. This idea is formalized in the next two definitions.

DEFINITION 2.8. *A* (Stackelberg) strategy *for the Stackelberg instance (M, r, α) is an assignment feasible for $(M, \alpha r)$.*

DEFINITION 2.9. *Let s be a strategy for Stackelberg instance (M, r, α) where machine $i \in M$ has latency function ℓ_i, and let $\tilde{\ell}_i(x_i) = \ell_i(s_i + x_i)$ for each $i \in M$. An* equilibrium induced by strategy *s is an assignment t at Nash equilibrium for the instance $(M, (1-\alpha)r)$ with respect to latency functions $\tilde{\ell}$. We then say that $s + t$ is an* assignment induced by *s for (M, r, α).*

Existence and essential uniqueness of induced equilibria follow easily from Lemmas 2.2 and 2.4.

LEMMA 2.10. *Let s be a strategy for a Stackelberg instance with continuous, nondecreasing latency functions. Then there exists an assignment induced by s, and any two such induced assignments have equal cost.*

The following simple observation will be useful in sections 4 and 5.

LEMMA 2.11. *Let s be a strategy for Stackelberg instance (M, r, α) inducing equilibrium t. Let M' denote the machines on which $t_i > 0$. Then $s + t$, restricted to M', is an assignment at Nash equilibrium for the instance $(M', s(M') + t(M'))$. In particular, all machines on which $t_i > 0$ have a common latency with respect to $s+t$.*

We next consider two examples that demonstrate both the usefulness and the limitations of Stackelberg strategies.

Example 2.12. Recall that in Example 2.3, with two machines with latency functions $\ell_1(x_1) = 1$ and $\ell_2(x_2) = x_2$, in the absence of centrally controlled jobs the

assignment at Nash equilibrium incurs total latency $\frac{4}{3}$ times that of the optimal assignment. Suppose instead that half of the jobs are controlled by the system manager (i.e., that $\alpha = \frac{1}{2}$) and consider the strategy $s = (\frac{1}{2}, 0)$. Then, as all remaining jobs will be assigned to the second machine in the equilibrium induced by s, the assignment induced by s is precisely the optimal assignment of all of the jobs. Thus, in this particular instance, system performance can be optimized via a Stackelberg strategy.

Example 2.13. Now modify Example 2.12 by replacing the latency function of the second machine with the latency function $\ell_2(x_2) = 2x_2$. The assignment at Nash equilibrium puts half of the jobs on each machine (for a cost of 1) while the optimal assignment is $(\frac{3}{4}, \frac{1}{4})$ (with a cost of $\frac{7}{8}$). On the other hand, if we again allow the system manager to assign half of the jobs, we see that for *any* strategy s, the assignment induced by s is $(\frac{1}{2}, \frac{1}{2})$ and hence is not optimal. In this example, there is no available strategy by which the system manager can improve system performance.

3. Three Stackelberg strategies.

3.1. Two natural strategies. We initiate our investigation of Stackelberg strategies by considering two natural approaches that provide suboptimal performance guarantees. To motivate our results in the simplest possible way, throughout this subsection we will consider examples in which all latency functions are linear and half of the jobs are centrally controlled ($\alpha = \frac{1}{2}$). We are thus hoping for strategies that always induce an assignment of cost at most $\frac{8}{7}$ times that of the optimal assignment (this is the best possible by Example 2.13).

First consider the following strategy for an instance $(M, r, \frac{1}{2})$: if x^* is the optimal assignment for instance $(M, \frac{1}{2}r)$, put $s = x^*$. In words, we choose the strategy of minimum cost, ignoring the existence of jobs that are not centrally controlled. We call this the *aloof* strategy since it refuses to acknowledge the rest of the jobs in the system. Example 2.3 shows that this strategy performs quite poorly: the strategy is $(0, \frac{1}{2})$ and the induced assignment is $(0, 1)$, an assignment that we have seen to incur total latency $\frac{4}{3}$ times that of the optimal assignment.

A second attempt for a good strategy might be as follows: if x^* is the optimal assignment for (M, r), put $s = \frac{1}{2}x^*$. We call this the *scale* strategy, since it is simply the optimal assignment of all the jobs, suitably scaled. Unfortunately, a simple example shows that the scale strategy also fails to provide the performance guarantee of $\frac{8}{7}$ that we are looking for: in a two-machine example with latency functions $\ell_1(x_1) = 1$ and $\ell_2(x_2) = \frac{3}{2}x_2$ and rate 1, the optimal assignment is $(\frac{2}{3}, \frac{1}{3})$ (with total cost $\frac{5}{6}$), and thus the scale strategy will be $(\frac{1}{3}, \frac{1}{6})$, which induces the assignment $(\frac{1}{3}, \frac{2}{3})$ having cost 1. Hence, the scale strategy may result in an induced assignment with total latency $\frac{6}{5}$ times the cost of the optimal assignment.[3]

3.2. The largest latency first (LLF) strategy. Intuitively, both the aloof and scale strategies suffer from a common flaw: both allocate jobs to machines that will subsequently be inundated in any induced equilibrium while assigning too little work to machines that selfish users are prone to ignore. This observation suggests that a good strategy should give priority to the machines that are least appealing to selfish users—machines with relatively high latency. With this intuition in mind, the following strategy for a Stackelberg instance (M, r, α), which we call the *largest latency first* (LLF) strategy, should seem natural:

(1) Compute the optimal assignment x^* for (M, r).

[3] In addition, the aloof and scale strategies can perform arbitrarily badly for instances with general latency functions.

(2) Index the machines of M so that $\ell_1(x_1^*) \leq \cdots \leq \ell_m(x_m^*)$.
(3) Let $k \leq m$ be minimal with $\sum_{i=k+1}^{m} x_i^* \leq \alpha r$.
(4) Put $s_i = x_i^*$ for $i > k$, $s_k = \alpha r - \sum_{i=k+1}^{m} x_i^*$, and $s_i = 0$ for $i < k$.

We will say that a machine i is *saturated* by a strategy s if $s_i = x_i^*$. The LLF strategy thus saturates machines one by one, in order from the largest latency with respect to x^* to the smallest, until there are no centrally controlled jobs remaining. Note that Lemma 2.5 implies that the LLF strategy can be computed in polynomial time (the bottleneck is step (1)); in section 5 we will see that it can be computed in $O(m^2)$ time when every latency function is linear.

The next two sections are devoted to proving that the LLF strategy always induces an assignment with near-optimal total latency.

4. A $\frac{1}{\alpha}$ performance guarantee for arbitrary latency functions. In this section we prove that the LLF strategy induces a near-optimal assignment for any set of latency functions and any number of machines. We note that *no* performance guarantee is possible in the absence of centrally controlled jobs: without additional restrictions on machine latency functions, the Nash assignment may incur arbitrarily more latency than the optimal assignment [45]. Thus, the benefit of a leader (and of a carefully chosen leader strategy) is particularly striking in this general setting.

A simple variation on previous examples demonstrates the limits of Stackelberg strategies. In a two-machine instance with $\alpha = \frac{1}{2}$ and latency functions $\ell_1(x_1) = 1$ and $\ell_2(x_2) = 2^k x_2^k$ for $k \in \mathcal{Z}^+$, any Stackelberg strategy induces the assignment $(\frac{1}{2}, \frac{1}{2})$ (having total latency 1) while the optimal assignment is $(\frac{1}{2} + \delta_k, \frac{1}{2} - \delta_k)$ having cost $\frac{1}{2} + \epsilon_k$, where $\delta_k, \epsilon_k \to 0$ as $k \to \infty$. Thus the best induced assignment may be (arbitrarily close to) twice as costly as the optimal assignment. Similar examples show that for any $\alpha \in (0, 1)$, the best induced assignment may be $\frac{1}{\alpha}$ times as costly as the optimal assignment.

The main result of this section is that the LLF strategy always induces an assignment of cost no more than $\frac{1}{\alpha}$ times that of the optimal assignment. A rough outline of the proof is as follows. Our goal is to exploit the iterative structure of the LLF strategy and proceed by induction on the number of machines. If the LLF strategy first saturates the mth machine, a natural idea is to apply the inductive hypothesis to the remainder of the LLF strategy on the first $m-1$ machines to derive a performance guarantee. This idea nearly succeeds, but there are two difficulties. First, it is possible that the LLF strategy fails to saturate any machines; we will see below that this case is easy to analyze and causes no trouble. Second, in order to obtain a clean application of the inductive hypothesis to the first $m-1$ machines, we require that the optimal and LLF-induced assignments place the same total amount of jobs on these machines—i.e., that the LLF-induced equilibrium eschews the mth machine.[4] We resolve this difficulty with the following lemma, which states that if the LLF strategy saturates the mth machine, then *some* induced equilibrium assigns all jobs to the first $m-1$ machines; this suffices for our purposes, since different induced assignments have equal cost.

LEMMA 4.1. *Let (M, r, α) denote a Stackelberg instance with optimal assignment x^* and index the machines of M so that $\ell_m(x_m^*) \geq \ell_i(x_i^*)$ for all i. If s is a strategy with $s_m = x_m^*$, then there exists an induced equilibrium t with $t_m = 0$.*

[4] To see a trivial example in which this does not occur, put $r = 1, \alpha = \frac{1}{2}$ and consider two machines each with the constant latency function $\ell_i(x_i) = 1$. One particular optimal assignment is $(\frac{2}{3}, \frac{1}{3})$, and a corresponding LLF strategy is $(\frac{1}{6}, \frac{1}{3})$; one particular induced assignment is $(\frac{1}{3}, \frac{2}{3})$. Even though the LLF strategy saturated the second machine, the induced equilibrium uses it.

Proof. Consider an arbitrary induced equilibrium t and suppose $t_m > 0$. Roughly speaking, the idea is to prove that this scenario occurs only when several latency functions (that of the mth machine, and others) are locally constant; then jobs assigned to machine m in the induced equilibrium can be evacuated to other machines with locally constant latency functions to provide a new induced equilibrium.

Formally, let $L = \ell_m(s_m + t_m) = \ell_m(x^*_m + t_m)$ denote the common latency with respect to $s + t$ of every machine with $t_i > 0$ (see Lemma 2.11). We must have $\ell_m(x^*_m) \geq L$; otherwise $\ell_i(x^*_i) < L$ for all i, yet $\ell_i(s_i + t_i) \geq L$ for all i, contradicting that x^* and $s + t$ are assignments at the same rate. Thus, since ℓ_m is nondecreasing, ℓ_m is locally constant: ℓ_m is equal to L in the interval $[x^*_m, x^*_m + t_m]$.

Next, let M' denote the machines on which $s_i + t_i < x^*_i$; since $s_m + t_m > x^*_m$, M' is nonempty. For each $i \in M'$, we know that $\ell_i(s_i + t_i) \geq L$, $\ell_i(x^*_i) \leq \ell_m(x^*_m) = L$, and ℓ_i is nondecreasing, so ℓ_i is equal to L on $[s_i + t_i, x^*_i]$. Since x^* and $s + t$ are assignments at the same rate, we must have $\sum_{i \in M'}[x^*_i - (s_i + t_i)] \geq t_m$. Finally, consider modifying t as follows: move all jobs previously assigned to machine m to machines in M', subject to the constraints $s_i + t_i \leq x^*_i$. We have already observed that there is sufficient "room" on machines in M' for this operation, and that all latency functions are constant in the domain of our modifications. We have thus exhibited a new induced equilibrium with no jobs assigned to machine m, completing the proof. ∎

We are now prepared to prove the main result of this section.

THEOREM 4.2. *Let $\mathcal{I} = (M, r, \alpha)$ denote a Stackelberg instance. If s is an LLF strategy for $\mathcal{I}$ inducing equilibrium t and x^* is an optimal assignment for the instance (M, r), then $C(s + t) \leq \frac{1}{\alpha} C(x^*)$.*

Proof. We proceed by induction on the number of machines m (for each fixed m, we will prove the theorem for arbitrary ℓ, r, and α). The case of one machine is trivial.

Fix a Stackelberg instance $\mathcal{I} = (M, r, \alpha)$ with at least two machines, and let x^* denote an optimal assignment to the instance (M, r) and s the corresponding LLF strategy. Index the machines so that $\ell_1(x^*_1) \leq \ell_2(x^*_2) \leq \cdots \leq \ell_m(x^*_m)$. By scaling, we may assume that $r = 1$ (use latency functions $\tilde{\ell}$ with $\tilde{\ell}_i(x_i) = \ell_i(r x_i)$). Let L denote the common latency with respect to $s + t$ of every machine with $t_i > 0$ (see Lemma 2.11).

Case 1. Suppose $t_k = 0$ for some machine k. Let M_1 denote the machines i for which $t_i = 0$ and M_2 the machines for which $t_i > 0$; both of these sets are nonempty. For $i = 1, 2$ let α_i denote the amount of centrally controlled jobs assigned to machines in M_i (so $\alpha_i = s(M_i)$) and C_i the cost incurred by $s + t$ on machines in M_i. By Lemma 2.11, $C_2 = (1 - \alpha_1)L$ and $C_1 \geq \alpha_1 L$. Further, since x^* restricted to M_2 is an optimal assignment for $(M_2, 1 - \alpha_1)$, s restricted to M_2 is an LLF strategy for the instance $\mathcal{I}_2 = (M_2, 1 - \alpha_1, \alpha')$, where $\alpha' = \frac{\alpha_2}{1 - \alpha_1}$.

Applying the inductive hypothesis to $\mathcal{I}_2$ and using the fact that $x^*_i \geq s_i = s_i + t_i$ for all $i \in M_1$, we obtain

$$C(x^*) \geq C_1 + \alpha' C_2.$$

Proving that $C(s + t) \leq \frac{1}{\alpha} C(x^*)$ thus reduces to showing

$$\alpha(C_1 + C_2) \leq C_1 + \alpha' C_2.$$

Since $\alpha \leq 1$ and $C_1 \geq \alpha_1 L$, it suffices to prove this inequality with C_1 replaced by $\alpha_1 L$. Writing $C_2 = (1 - \alpha_1)L$ and $\alpha' = \frac{\alpha_2}{1 - \alpha_1}$ and dividing through by L, we need

only check that

$$\alpha(\alpha_1 + (1-\alpha_1)) \le \alpha_1 + \frac{\alpha_2}{1-\alpha_1}(1-\alpha_1),$$

which clearly holds (both sides are equal to α).

Case 2. Suppose $t_i > 0$ for every machine i, so $C(s+t) = L$. We may assume that the LLF strategy failed to saturate machine m; otherwise, by Lemma 2.10, we can finish by applying the previous case to the better-behaved induced assignment guaranteed by Lemma 4.1. Thus, $\alpha < x^*_m$.

As in the proof of Lemma 4.1, we must have $\ell_m(x^*_m) \ge L$; otherwise, $\ell_i(x^*_i) < L$ for all machines i, while $\ell_i(s_i + t_i) = L$ for all i, contradicting that x^* and $s+t$ are assignments at the same rate. Having established that machine m has large latency with respect to x^* and that x^*_m is fairly large, it is now a simple matter to lower bound $C(x^*)$:

$$C(x^*) \ge x^*_m \ell_m(x^*_m) \ge \alpha L = \alpha C(s+t). \qquad \square$$

5. A $\frac{4}{3+\alpha}$ performance guarantee for linear latency functions.

5.1. Properties of the Nash and optimal assignments. In this subsection we undertake a deeper study of the Nash and optimal assignments for instances with linear latency functions. The results of this subsection will be instrumental in proving a stronger performance guarantee for the LLF strategy for these instances.

Fix a set of machines M with latency functions $\ell_i(x_i) = a_i x_i + b_i$ for each i ($a_i, b_i \ge 0$) and index them so that $b_1 \le b_2 \le \cdots \le b_m$. We may assume that at most one machine has a constant latency function ($a_i = 0$) since all but the fastest may be safely discarded; under this assumption, the Nash and optimal assignments are always unique. We may similarly assume that a machine with a constant latency function is the last machine.

Our first goal is to understand the structure of the Nash assignment $\bar{x}$ as a function of the rate r. It is useful to imagine r increasing from 0 to a large value, with the corresponding Nash assignment changing in a continuous fashion; an intuitive description of this process is as follows. Initially, when r is nearly zero, all jobs will be assigned to the machine having the smallest constant term. Once the first machine is sufficiently loaded, the second machine looks equally attractive. This occurs when $a_1\bar{x}_1 + b_1 = b_2$—when the load on machine 1 is $\frac{b_2 - b_1}{a_1}$. At this point, new jobs will be assigned to both of the first two machines, at rates proportional to $\frac{1}{a_1}$ and $\frac{1}{a_2}$ so that these two machines continue to have equal latency. Once $(b_3 - b_2)(\frac{1}{a_1} + \frac{1}{a_2})$ further units of work have arrived and been assigned to the first two machines, machine 3 will be equally attractive and new jobs will be spread out among the first three machines, and so on. We may thus envision the Nash assignment as being constructed in phases: within phase i jobs are assigned to the first i machines according to fixed relative proportions and at the end of the phase, after enough new jobs have been assigned, an additional machine is put into use.

We now formalize this intuitive description of the Nash assignment $\bar{x}$. For $i = 1, \ldots, m$, let v_i denote the m-vector $(\frac{1}{a_1}, \frac{1}{a_2}, \ldots, \frac{1}{a_i}, 0, 0, \ldots, 0) \in \mathcal{R}^m_+$; if $a_m = 0$, put $v_m = (0, 0, \ldots, 1)$. The vector v_i should be interpreted as a specification of the way jobs are assigned to the first i machines during the ith phase. Next, define δ_i for $i = 0, 1, \ldots, m-1$ inductively by $\delta_0 = 0$ and $\delta_i = \min\{(b_{i+1} - b_i)\|v_i\|_1, r - \sum_{j=0}^{i-1} \delta_i\} \ge 0$, where $\|\cdot\|_1$ denotes the L_1-norm. We also put $\delta_m = r - \sum_{j=0}^{m-1} \delta_i$. The scalar δ_i is the amount of jobs assigned in the ith phase. We can then describe $\bar{x}$ as follows.

LEMMA 5.1. *Let $\mathcal{I}$ be an instance with linear latency functions, as above. Then the Nash assignment for $\mathcal{I}$ is given by*

$$\bar{x} = \sum_{i=1}^{m} \delta_i \frac{v_i}{\|v_i\|_1}.$$

Our characterization of optimal assignments (Lemma 2.5 and Corollary 2.6) yields an analogous result for computing them by an explicit formula. Note that when a latency function has the form $\ell_i(x_i) = a_i x_i + b_i$, the corresponding marginal cost function is $\ell_i^*(x_i) = 2a_i x_i + b_i$. Recalling from Corollary 2.6 that an optimal assignment is simply a Nash assignment with respect to latency functions ℓ^*, we see that the optimal assignment is created by the same process as the Nash assignment, except that new machines are incorporated at a more rapid pace so as to spread jobs over a wider range of machines and thus achieve a smaller total latency.

Formally, let v_i be as above and define δ_i^* inductively by $\delta_0^* = 0$, $\delta_i^* = \min\{\frac{1}{2}(b_{i+1} - b_i)\|v_i\|_1, r - \sum_{j=0}^{i-1} \delta_i^*\}$, and $\delta_m^* = r - \sum_{j=0}^{m-1} \delta_i^*$. Letting x^* denote the optimal assignment to (M, r), the analogue of Lemma 5.1 is as follows.

LEMMA 5.2. *Let $\mathcal{I}$ be an instance with linear latency functions, as above. Then the optimal assignment for $\mathcal{I}$ is given by*

$$x^* = \sum_{i=1}^{m} \delta_i^* \frac{v_i}{\|v_i\|_1}.$$

Lemmas 5.1 and 5.2 have several useful corollaries. We summarize them below.

COROLLARY 5.3. *Let M be a set of machines with latency functions $\{\ell_i(x_i) = a_i x_i + b_i\}_{i \in M}$ and at most one machine with a constant latency function. Let b_i be nondecreasing in i. Then*

(a) *if x^* and $\bar{x}$ denote the optimal and Nash assignments to (M, r), then $x_m^* \geq \bar{x}_m$;*

(b) *if x^* and $\bar{x}$ denote the optimal and Nash assignments to (M, r), then $x_1^* \leq \bar{x}_1 \leq 2x_1^*$;*

(c) *if x^* is the optimal assignment for (M, r_1) and y^* is the optimal assignment for (M, r_2) with $r_1 \geq r_2$, then $x_i^* \geq y_i^*$ for each i;*

(d) *for any rate r, the optimal and Nash assignments of (M, r) can be computed in $O(m^2)$ time.*

Proof. Parts (c) and (d) are immediate from Lemmas 5.1 and 5.2. For the remaining parts, fix an instance (M, r) and define v_i, δ_i, and δ_i^* as in Lemmas 5.1 and 5.2. Our first observation is that, for any $i \in \{1, 2, \ldots, m\}$, the Nash assignment schedules at least as many jobs in the first i phases as the optimal assignment—formally, that $\sum_{k=1}^{i} \delta_k \geq \sum_{k=1}^{i} \delta_k^*$ for all i. Since $\sum_{k=1}^{m} \delta_k = \sum_{k=1}^{m} \delta_k^* = r$, we obtain $\delta_m^* \geq \delta_m$ and hence $x_m^* \geq x_m$, proving (a).

It remains to prove part (b) of the corollary. We may assume that $m > 1$ (otherwise $\bar{x}_1 = x_1^* = r$). Letting m' equal m if there is no machine with constant latency function and $m - 1$ otherwise, Lemmas 5.1 and 5.2 give

$$x_1^* = \frac{1}{a_1} \sum_{i=1}^{m'} \frac{\delta_i^*}{\|v_i\|_1}$$

and

$$\bar{x}_1 = \frac{1}{a_1} \sum_{i=1}^{m'} \frac{\delta_i}{\|v_i\|_1}.$$

By the definitions of δ_i and δ_i^*, we have $\delta_i \le 2\delta_i^*$ for $i = 1, 2, \ldots, m$ and hence $\bar{x}_1 \le 2x_1^*$. For the other inequality, we recall that $\sum_{k=1}^{i} \delta_k^* \le \sum_{k=1}^{i} \delta_k$ for each i and observe that $\|v_i\|_1$ is increasing in i (for $i \in \{1, 2, \ldots, m'\}$); it follows that $x_1^* \le \bar{x}_1$. □

Corollary 5.3(d) implies that the LLF strategy can be computed in $O(m^2)$ time for instances with linear latency functions.

5.2. Proof of performance guarantee. In subsection 2.3 we saw an example with linear latency functions and $\alpha = \frac{1}{2}$ in which *no* strategy can induce an assignment with cost less than $\frac{8}{7}$ times that of the optimal assignment. This example is easily modified to show that, for any $\alpha \in (0, 1)$, the minimum-cost induced assignment for a Stackelberg instance (M, r, α) with linear latency functions may be $\frac{4}{3+\alpha}$ times as costly as the optimal assignment for (M, r). The main result of this section is a matching upper bound for the LLF strategy.

Before proving this result, we give an alternative description of LLF that is more convenient for our analysis. This description is based on the following lemma.

LEMMA 5.4. *Let x^* be an optimal assignment for (M, r) where machine i has latency function $\ell_i(x_i) = a_i x_i + b_i$. Then $\ell_i(x_i^*) \ge \ell_j(x_j^*)$ if and only if $b_i \ge b_j$.*

Proof. The lemma is clear when $x_i^* = x_j^* = 0$. Otherwise, we will make use of our characterization of optimal assignments via machine marginal cost functions (Lemma 2.5). If exactly one of x_i^*, x_j^* is 0 (say x_i^*), then by Lemma 2.5 we know that the marginal cost $\ell_i^*(x_i^*) = \ell_i^*(0) = b_i$ of machine i is at least the marginal cost $\ell_j^*(x_j^*) = 2a_j x_j^* + b_j$ of machine j. Thus we necessarily have both $\ell_i(x_i^*) \ge \ell_j(x_j^*)$ and $b_i \ge b_j$. Finally, if $x_i^*, x_j^* > 0$, then by Lemma 2.5 we have $2a_i x_i^* + b_i = 2a_j x_j^* + b_j = L^*$ for some L^*; thus $b_i \ge b_j$ if and only if $a_i x_i^* \le a_j x_j^*$. The lemma follows by writing $\ell_i(x_i^*) = L^* - a_i x_i^*$ and $\ell_j(x_j^*) = L^* - a_j x_j^*$. □

Lemma 5.4 gives the following equivalent description of the LLF strategy: saturate machines one by one, in decreasing order of constant terms, until no centrally controlled jobs remain. It may seem surprising that the LLF strategy makes no use of the a_i-values in ordering the machines; however, this is consistent with our observation in subsection 5.1 that the order in which machines are used by the optimal assignment, if we think of the rate as increasing from 0 to some large value, depends only on the constant terms of the machines' latency functions.

We are finally prepared to prove a $\frac{4}{3+\alpha}$ performance guarantee for the LLF strategy for instances with linear latency functions. The general approach is similar to that of Theorem 4.2 and is again by induction on the number of machines. However, new difficulties arise in proving a stronger performance guarantee. The case in which there is some machine k on which the induced equilibrium assigns no jobs (i.e., $t_k = 0$ for some k) is nearly identical to the first case of Theorem 4.2: the desired performance guarantee can easily be extracted from the inductive guarantee for the smaller instance of machines on which $t_i > 0$. The second case, in which the induced equilibrium assigns jobs to all machines, is substantially more complicated. In particular, the simple approach in the proof of Theorem 4.2 does *not* use any inductive guarantee in this case and is thus not strong enough to prove a guarantee better than $\frac{1}{\alpha}$. For this reason, much of the proof is devoted to defining an appropriate smaller instance that allows for clean application of the inductive hypothesis and for extending the inductive guarantee into one for the original instance.

THEOREM 5.5. *Let $\mathcal{I} = (M, r, \alpha)$ denote a Stackelberg instance with linear latency functions. If s is an LLF strategy for $\mathcal{I}$ inducing equilibrium t and x^* is an optimal assignment for (M, r), then $C(s + t) \le \frac{4}{3+\alpha} C(x^*)$.*

Proof. We proceed by induction on the number of machines m (for each fixed m,

we will prove the theorem for arbitrary (linear) ℓ, r, and α). The case of one machine is trivial.

Fix a Stackelberg instance $\mathcal{I} = (M, r, \alpha)$ with at least two machines and let $\ell_i(x_i) = a_i x_i + b_i$ (with $a_i, b_i \geq 0$). Let x^* denote an optimal assignment to (M, r). We begin with several simplifying assumptions, each made with no loss of generality. As in Theorem 4.2, we may assume that $r = 1$. We assume as usual that there is at most one machine with a constant latency function. It will also be convenient to assume that some machine i has constant term 0. To enforce this assumption we may subtract $\min_i b_i$ from every latency function before applying our argument: assuming $r = 1$, this modification decreases the cost of every assignment by precisely $\min_i b_i$ and will only increase the ratio in costs between any two assignments. Finally, we assume that no machine has latency function $\ell_i(x_i) = 0$ (otherwise the instance is trivial).

Let s denote an LLF strategy for $\mathcal{I}$ and t the induced equilibrium. Let $L > 0$ denote the common latency of every machine used by t (see Lemma 2.11). Index the machines of M as in the second description of the LLF strategy (see Lemma 5.4), so that $0 = b_1 \leq b_2 \leq \cdots \leq b_m$ and $a_1 > 0$. We will need to apply the inductive hypothesis in two different ways, and our analysis breaks into two cases.

Case 1. Suppose $t_k = 0$ for some k. Our argument will be essentially identical to the first case in the proof of Theorem 4.2. Let M_1 denote the machines on which $t_i = 0$ and M_2 the machines on which $t_i > 0$. For $i = 1, 2$, let α_i denote the amount of centrally controlled jobs on machines in M_i. For $i = 1, 2$ let C_i denote the cost incurred by $s + t$ on machines in M_i. Observe that $C_1 \geq \alpha_1 L$ and $C_2 = (1 - \alpha_1)L$. Since x^* restricted to M_2 is an optimal assignment for $(M_2, 1 - \alpha_1)$, s restricted to M_2 is an LLF strategy for $\mathcal{I}_2 = (M_2, 1 - \alpha_1, \alpha')$, where $\alpha' = \frac{\alpha_2}{1-\alpha_1}$. The inductive hypothesis, applied to $\mathcal{I}_2$, and the fact that $x^*_i \geq s_i = s_i + t_i$ for all $i \in M_1$ imply that

$$C(x^*) \geq C_1 + \frac{3 + \alpha'}{4} C_2.$$

Proving that $C(s + t) \leq \frac{4}{3+\alpha} C(x^*)$ thus reduces to showing

$$(3 + \alpha)(C_1 + C_2) \leq 4C_1 + (3 + \alpha')C_2.$$

Since $\alpha \leq 1$ and $C_1 \geq \alpha_1 L$, it suffices to prove this inequality with C_1 replaced by $\alpha_1 L$. Writing $C_2 = (1 - \alpha_1)L$, $\alpha' = \frac{\alpha_2}{1-\alpha_1}$, and dividing through by L verifies the result.

Case 2. Suppose $t_i > 0$ for all machines $i \in M$. This implies that $s + t$ is a Nash assignment for $(M, 1)$; by Corollary 5.3(a) we have $s_m < s_m + t_m \leq x^*_m$ (in particular, we must have $x^*_m > 0$). It follows that the LLF strategy s failed to saturate machine m, so $s_m = \alpha$ and $s_i = 0$ for $i < m$.

Our first goal is to show that s is an LLF strategy not only for $\mathcal{I}$ but also for $\mathcal{I}' = (M', 1 - t_1, \frac{\alpha}{1-t_1})$, where $M' = M \setminus \{1\}$ (we may then apply the inductive hypothesis to s in this smaller instance). Toward this end, let y^* denote the optimal assignment to the instance $(M', 1 - t_1)$. Since $s + t$ restricted to M' is a Nash assignment for $(M', 1 - t_1)$, we must have $y^*_m \geq s_m + t_m$ (see Corollary 5.3(a)); since $\alpha = s_m < s_m + t_m \leq y^*_m$, the LLF strategy for $\mathcal{I}'$ is precisely s (restricted to M').

Let C^*_1, C^*_2 denote the total latency incurred by x^* on machine 1 and in M', respectively. The next claim gives a lower bound on C^*_2, as a function of the amount of jobs assigned to machines in M' in the optimal assignment.

CLAIM. *If $r' \geq 1 - t_1$, then the cost of the optimal assignment for (M', r') is at least*

$$\frac{3+\alpha'}{4}(1-t_1)L + (r' - 1 + t_1)L,$$

where $\alpha' = \frac{\alpha}{1-t_1}$.

Proof. The claim is proved for $r' = 1 - t_1$ by applying the inductive hypothesis to the instance $\mathcal{I}' = (M', 1 - t_1, \alpha')$ and using the fact that s is an LLF strategy for M' inducing an assignment of cost $(1-t_1)L$. Suppose now that $r' > 1-t_1$. We again denote the optimal assignment for $(M', 1 - t_1)$ by y^*. Since y^* and $s + t$ (restricted to M') are assignments at the same rate (namely, $1 - t_1$) and the common latency of every machine with respect to $s + t$ is L, there is some machine i with $y_i^* > 0$ and $\ell_i(y_i^*) \geq L$. Since the marginal cost of a machine is at least its latency, Lemma 2.5 implies that the marginal cost of every machine in M' is at least L with respect to y^*. By Corollary 5.3(c) and the observation that marginal costs are nondecreasing functions of the machine load, extending y^* from an optimal assignment for $(M', 1-t_1)$ to an optimal assignment for (M', r') involves the assignment of $r' - (1 - t_1)$ units of jobs, all assigned at a marginal cost of at least L. Thus, the overall cost of an optimal assignment to (M', r') must be at least $C(y^*) + (r' - 1 + t_1)L \geq \frac{3+\alpha'}{4}(1 - t_1)L + (r' - 1 + t_1)L$. ∎

Our goal is to prove that $(3 + \alpha)C(s + t) \leq 4C(x^*) = 4(C_1^* + C_2^*)$; with the claim in hand, we have reduced the proof of the theorem to proving the inequality

$$(3+\alpha)L \leq (3+\alpha')(1-t_1)L + 4(t_1 - x_1^*)L + 4a_1(x_1^*)^2,$$

where $\alpha' = \frac{\alpha}{1-t_1}$ (recall that $\ell_1(x_1) = a_1 x_1$ and hence $C_1^* = a_1(x_1^*)^2$). For any fixed value of t_1, $x_1^* \in [\frac{1}{2}t_1, t_1]$ (see Corollary 5.3(b)). Using the identity $a_1 t_1 = L$ and differentiating, we find that the right-hand side is minimized by $x_1^* = \frac{1}{2}t_1$. Since the left-hand side is independent of x_1^*, it suffices to prove that

$$(3+\alpha)L \leq (3+\alpha')(1-t_1)L + 2t_1 L + a_1 t_1^2.$$

Substituting for α', using the identity $a_1 t_1 = L$, and dividing by L gives

$$3 + \alpha \leq \left(3 + \frac{\alpha}{1-t_1}\right)(1-t_1) + 3t_1,$$

which clearly holds, proving the theorem. ∎

6. The complexity of computing optimal strategies. Thus far, we have measured the performance of a Stackelberg strategy by comparing the cost of the corresponding induced assignment to the cost of the optimal assignment of all of the jobs. Another natural approach for evaluating a strategy is to compare the cost of the induced assignment to that of the *least costly assignment induced by some Stackelberg strategy*, i.e., to the cost of the assignment induced by the *optimal* strategy. Motivated by the latter measure, in this section we study the optimization problem of computing the optimal Stackelberg strategy.

We have seen that the LLF strategy provides the best possible (worst-case) performance guarantee relative to the cost of the optimal assignment, and in particular that the algorithm of subsection 3.2 may be viewed as a $\frac{1}{\alpha}$-approximation algorithm[5]

[5] A *c-approximation algorithm* for a minimization problem runs in polynomial time and returns a solution no more than c times as costly as an optimal solution. The value c is the *approximation ratio* or *performance guarantee* of the algorithm.

for computing the optimal strategy, and a $\frac{4}{3+\alpha}$-approximation algorithm for instances with linear latency functions. However, simple examples show that the LLF strategy is *not* always the optimal strategy, and thus our algorithm fails to solve this optimization problem exactly.[6] Our main result in this section is strong evidence that no such polynomial-time algorithm exists.

THEOREM 6.1. *The problem of computing the optimal Stackelberg strategy is NP-hard, even for instances with linear latency functions.*

Proof. We reduce from a problem we call $\frac{1}{3}$-$\frac{2}{3}$ PARTITION: given n positive integers $a_1, a_2, \ldots, a_n$, is there a subset $S \subseteq \{1, 2, \ldots, n\}$ satisfying $\sum_{i \in S} a_i = \frac{1}{3} \sum_{i=1}^{n} a_i$? The canonical reduction from the NP-complete problem SUBSET SUM to PARTITION is easily modified to show that $\frac{1}{3}$-$\frac{2}{3}$ PARTITION is NP-hard; see Garey and Johnson [21] for problem definitions and Karp [25] or Kozen [31, p. 129] for the canonical reduction. We will show that deciding the problem $\frac{1}{3}$-$\frac{2}{3}$ PARTITION reduces to deciding whether or not a given Stackelberg scheduling instance with linear latency functions admits a Stackelberg strategy inducing an assignment with a given cost.

Given an arbitrary instance $\mathcal{I}$ of $\frac{1}{3}$-$\frac{2}{3}$ PARTITION specified by positive integers $a_1, \ldots, a_n$, put $A = \sum_{i=1}^{n} a_i$ and define a Stackelberg scheduling instance $\mathcal{I}' = (\{1, 2, \ldots, n+1\}, 2A, \frac{1}{4})$ with $n+1$ machines and with linear latency functions $\ell_i(x_i) = \frac{x_i}{a_i} + 4$ for $i = 1, \ldots, n$ and $\ell_{n+1}(x_{n+1}) = \frac{3x_{n+1}}{A}$. It is clear that $\mathcal{I}'$ can be constructed from $\mathcal{I}$ in polynomial time. We claim that $\mathcal{I}$ is a "yes instance" (that is, admits a $\frac{1}{3}$-$\frac{2}{3}$ partition) if and only if there is a Stackelberg strategy for instance $\mathcal{I}'$ inducing an assignment with cost at most $\frac{35}{4}A$.

First suppose $\mathcal{I}$ is a "yes instance" of $\frac{1}{3}$-$\frac{2}{3}$ PARTITION, with $S \subseteq \{1, 2, \ldots, n\}$ satisfying $\sum_{i \in S} a_i = \frac{2}{3}A$, and consider the strategy defined by $s_i = \frac{3}{4}a_i$ for $i \in S$ and $s_i = 0$ otherwise (since $\sum_{i=1}^{n+1} s_i = \frac{3}{4} \sum_{i \in S} a_i = \frac{3}{4}\frac{2}{3}A = \frac{1}{2}A$, this defines a Stackelberg strategy). The induced equilibrium is then $t_i = 0$ for $i \in S$, $t_i = \frac{1}{4}a_i$ for $i \in \{1, 2, \ldots, n\} \setminus S$, and $t_{n+1} = \frac{17}{12}A$. In the induced assignment $s + t$, the $A/2$ units of jobs on machines corresponding to S experience $\frac{19}{4}$ units of latency, while the other $3A/2$ units of jobs experience $\frac{17}{4}$ units of latency. The cost of $s + t$ is thus

$$\frac{A}{2}\frac{19}{4} + \frac{3A}{2}\frac{17}{4} = \frac{35}{4}A.$$

Now suppose that $\mathcal{I}$ is a "no instance" of $\frac{1}{3}$-$\frac{2}{3}$ PARTITION, and consider any Stackelberg strategy s for $\mathcal{I}'$, inducing equilibrium t. We need to show that $C(s+t) > \frac{35}{4}A$. Call machine i *heavy* if $t_i = 0$ and *light* otherwise. Our first observation is that machine $n + 1$ must be light (even if all centrally controlled jobs are assigned to machine $n + 1$, some selfishly controlled jobs are subsequently assigned to it). Next, we note that for $i, j \in \{1, 2, \ldots, n\}$, the marginal cost $\frac{2(s_i+t_i)}{a_i} + 4$ of machine i is at most the marginal cost $\frac{2(s_j+t_j)}{a_j} + 4$ of machine j if and only if the latency $\ell_i(s_i + t_i)$ of machine i is at most $\ell_j(s_j + t_j)$. We may assume that all heavy machines have the same marginal cost with respect to $s + t$, as reassigning some centrally controlled jobs from a heavy machine with large marginal cost to a heavy machine with small marginal cost does not affect the induced equilibrium and can only decrease the cost of the induced assignment. All heavy machines must then have equal latency with

[6] For example, consider a three-machine instance with latency functions $\ell_1(x_1) = x_1$, $\ell_2(x_2) = 1 + x_2$, $\ell_3(x_3) = 1 + x_3$, with $r = 1$ and $\alpha = \frac{1}{6}$. The optimal assignment is $(\frac{2}{3}, \frac{1}{6}, \frac{1}{6})$ and thus the LLF strategy is $(0, 0, \frac{1}{6})$ inducing the assignment $(\frac{5}{6}, 0, \frac{1}{6})$ with cost $\frac{8}{9}$. On the other hand, the strategy $(0, \frac{1}{12}, \frac{1}{12})$ induces the assignment $(\frac{5}{6}, \frac{1}{12}, \frac{1}{12})$ having cost $\frac{7}{8}$.

respect to $s+t$. Naturally, Lemma 2.11 implies that all light machines possess a common latency with respect to $s+t$. That all machines have one of two latencies will make the cost of $s+t$ easy to compute.

With the induced assignment $s+t$ still fixed, let $S \subseteq \{1,2,\ldots,n\}$ denote the set of heavy machines. If $S=\emptyset$, then $s+t$ is the Nash assignment for $\mathcal{I}'$ with $s_i+t_i=\frac{a_i}{2}$ for $i \in \{1,2,\ldots,n\}$ and $s_{n+1}+t_{n+1}=\frac{3}{2}A$, satisfying $C(s+t)=9A>\frac{35}{4}A$. So suppose S is nonempty and define $\beta \in (0,1]$ by the equation $\sum_{i\in S} a_i = \beta A$. Define $\gamma \in (0,\frac{1}{2}]$ by the equation $\sum_{i\in S} s_i = \gamma A$. Our aim is to lower bound the cost of $s+t$ as a function of the parameters β and γ.

Since all heavy machines have equal latency, for i heavy we must have $s_i+t_i = s_i = \frac{\gamma}{\beta}a_i$ with $\ell_i(s_i+t_i)=4+\frac{\gamma}{\beta}$. Since all light machines have equal latency, we must have

$$s_i+t_i = a_i\frac{2-3\gamma}{4-3\beta}$$

with

$$\ell_i(s_i+t_i)=4+\frac{2-3\gamma}{4-3\beta}$$

for $i \in \{1,2,\ldots,n\}\setminus S$ and

$$s_{n+1}+t_{n+1}=A\left(\frac{4}{3}+\frac{2-3\gamma}{12-9\beta}\right)$$

with

$$\ell_{n+1}(s_{n+1}+t_{n+1})=4+\frac{2-3\gamma}{4-3\beta}.$$

The total cost of this solution is

$$\begin{aligned} C(s+t) &= \gamma A\left(4+\frac{\gamma}{\beta}\right)+(2-\gamma)A\left(4+\frac{2-3\gamma}{4-3\beta}\right) \\ &= A\left(8+\frac{(4-3\beta)\gamma^2+\beta(2-\gamma)(2-3\gamma)}{\beta(4-3\beta)}\right). \end{aligned}$$

Holding β fixed and differentiating with respect to γ, we find that this expression has unique minimizer $\gamma=\beta$ when $\beta \le \frac{1}{2}$ and $\gamma=\frac{1}{2}$ when $\beta \ge \frac{1}{2}$ (subject to the condition $\gamma \in (0,\frac{1}{2}]$). There are now two cases to analyze. First suppose that $\beta \le \frac{1}{2}$. Setting $\gamma=\beta$ we obtain

$$C(s+t)=A\left(8+\frac{4-4\beta}{4-3\beta}\right);$$

differentiating with respect to β, we see that the expression has a unique minimizer $\beta=\frac{1}{2}$ (subject to the condition $\beta \in (0,\frac{1}{2}]$) yielding cost $A(8+\frac{4}{5})>\frac{35}{4}A$. Finally, assume that $\beta \ge \frac{1}{2}$. Setting $\gamma=\frac{1}{2}$ we find that the cost of $s+t$ is given by

$$C(s+t)=A\left(8+\frac{1}{\beta(4-3\beta)}\right);$$

differentiating with respect to β, we find that this expression has unique minimizer $\beta=\frac{2}{3}$, at which point the equation reads $C(s+t)=\frac{35}{4}A$. However, since $\mathcal{I}$ is a "no instance" of $\frac{1}{3}$-$\frac{2}{3}$ PARTITION, we must have $\beta \ne \frac{2}{3}$ and hence $C(s+t)>\frac{35}{4}A$. We have exhausted all possible cases, and the reduction is complete. ∎

7. Directions for future work. The results of this paper suggest a number of problems deserving further study. An important and general open question is to what extent our machine-scheduling results carry over to the more complex domain of general networks. For example, given a directed graph G with a source vertex s and a sink t, a rate r of network traffic that wishes to travel from s to t, a latency function on each arc, and a parameter α specifying the fraction of traffic that is centrally controlled, how should the managed traffic be routed so as to induce the best possible equilibrium?[7] On the one hand, an example in a four-vertex network shows that a worst-case performance guarantee as small as $1/\alpha$ is impossible to attain with general latency functions [44]. On the other hand, the example does not rule out the possibility of a guarantee that is a larger function of α. Is it always possible to induce an assignment with cost no more than a constant—depending on α, but *not* on the size of the network—times that of the optimal assignment of traffic to s-t paths? Is a performance guarantee of $\frac{4}{3} - \epsilon$ possible for the special case of linear latency functions?

There are also unexplored avenues in the machine-scheduling setting. In section 6, we considered the optimization problem of computing the optimal Stackelberg strategy and observed that the LLF strategy achieves the best approximation ratio possible using the cost of the optimal assignment as a lower bound. Can a better approximation ratio be proved for LLF via a better lower bound on the cost of the assignment induced by the optimal strategy? In a different direction, to what extent do the results of this paper extend to systems with jobs of nonnegligible size?

Another natural question is whether more sophisticated approximation algorithms can achieve a better approximation ratio. This question has been resolved in the affirmative by Kumar and Marathe [32], who have given a fully polynomial-time approximation scheme[8] for the problem under mild conditions on the machine latency functions.

Acknowledgments. We thank Éva Tardos for several helpful discussions and for comments on an earlier draft of this paper. We also thank Anupam Gupta for useful discussions, Ben Atkin and Jon Kleinberg for pointing out relevant references, and the anonymous referees for their helpful comments.

REFERENCES

[1] E. Anshelevich, A. Dasgupta, É. Tardos, and T. Wexler, *Near-optimal network design with selfish agents*, in Proceedings of the 35th Annual ACM Symposium on the Theory of Computing, 2003, ACM Press, New York, pp. 511–520.

[2] A. Archer and É. Tardos, *Truthful mechanisms for one-parameter agents*, in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 482–491.

[3] A. Bagchi, *Stackelberg Differential Games in Economic Models*, Springer-Verlag, Berlin, 1984.

[4] T. Başar and G. J. Olsder, *Dynamic Noncooperative Game Theory*, SIAM, Philadelphia, 1999.

[5] M. Beckmann, C. B. McGuire, and C. B. Winsten, *Studies in the Economics of Transportation*, Yale University Press, New Haven, CT, 1956.

[7] The notions of equilibria used in this paper are equally easy to define in general networks; see, e.g., [45] for a formal treatment.

[8] A *fully polynomial-time approximation scheme* for a minimization problem is an algorithm A with the following property for some polynomial $p(\cdot,\cdot)$: given error parameter $\epsilon > 0$ and problem instance $\mathcal{I}$ with size $|\mathcal{I}|$, A returns a solution to $\mathcal{I}$ with objective function value at most $1+\epsilon$ times that of optimal in time at most $p(|\mathcal{I}|, \epsilon^{-1})$.

[6] K. P. Birman, *Building Secure and Reliable Network Applications*, Manning, Greenwich, CT, 1996.

[7] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. J. Shenker, J. Wroclawski, and L. Zhang, *Recommendations on Queue Management and Congestion Avoidance in the Internet*, Network Working Group Request for Comments 2309, 1998; available online from http://www.faqs.org/rfcs/rfc2309.html.

[8] R. Cocchi, S. J. Shenker, D. Estrin, and L. Zhang, *Pricing in computer networks: Motivation, formulation, and example*, IEEE/ACM Trans. Networking, 1 (1993), pp. 614–627.

[9] R. Cole, Y. Dodis, and T. Roughgarden, *How much can taxes help selfish routing?*, in Proceedings of the Fourth Annual ACM Conference on Electronic Commerce, ACM Press, New York, 2003, pp. 98–107.

[10] R. Cole, Y. Dodis, and T. Roughgarden, *Pricing network edges for heterogeneous selfish users*, in Proceedings of the 35th Annual ACM Symposium on the Theory of Computing, 2003, ACM Press, New York, pp. 521–530.

[11] A. Czumaj, P. Krysta, and B. Vöcking, *Selfish traffic allocation for server farms*, in Proceedings of the 34th Annual ACM Symposium on the Theory of Computing, ACM Press, New York, 2002, pp. 287–296.

[12] A. Czumaj and B. Voecking, *Tight bounds for worst-case equilibria*, in Proceedings of the 13th Annual Symposium on Discrete Algorithms, ACM Press, New York, 2002, pp. 413–420.

[13] S. C. Dafermos and F. T. Sparrow, *The traffic assignment problem for a general network*, J. Res. Nat. Bur. Standards Sect. B, 73 (1969), pp. 91–118.

[14] C. Douligeris and R. Mazumdar, *Multilevel flow control of queues*, in Proceedings of the Johns Hopkins Conference on Information Sciences and Systems, Johns Hopkins University Press, Baltimore, MD, 1989, p. 21.

[15] C. Douligeris and R. Mazumdar, *A game theoretic perspective to flow control in telecommunication networks*, J. Franklin Inst., 329 (1992), pp. 383–402.

[16] P. Dubey, *Inefficiency of Nash equilibria*, Math. Oper. Res., 11 (1986), pp. 1–8.

[17] N. G. Duffield, P. Goyal, A. G. Greenberg, P. P. Mishra, K. Ramakrishnan, and J. E. van der Merwe, *A flexible model for resource management in virtual private networks*, in Proceedings of ACM SIGCOMM, Comput. Commun. Rev. 29, ACM Press, New York, 1999, pp. 95–108.

[18] A. A. Economides and J. A. Silvester, *Priority load sharing: An approach using Stackelberg games*, in Proceedings of the 28th Annual Allerton Conference on Communications, Control, and Computing, 1990, pp. 674–683.

[19] J. Feigenbaum, C. H. Papadimitriou, and S. J. Shenker, *Sharing the cost of multicast transmissions*, J. Comput. System Sci., 63 (2001), pp. 21–41.

[20] J. A. Fingerhut, S. Suri, and J. S. Turner, *Designing least-cost nonblocking broadband networks*, J. Algorithms, 24 (1997), pp. 287–309.

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

[22] A. Gupta, A. Kumar, J. Kleinberg, R. Rastogi, and B. Yener, *Provisioning a virtual private network: A network design problem for multicommodity flow*, in Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing, ACM Press, New York, 2001, pp. 389–398.

[23] L. A. Hall, *Approximation algorithms for scheduling*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS, Boston, MA, 1997, ch. 1, pp. 1–45.

[24] R. Johari and J. N. Tsitsiklis, *Network Resource Allocation and a Congestion Game*, manuscript, 2003.

[25] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[26] Y. A. Korilis, A. A. Lazar, and A. Orda, *Achieving network optima using Stackelberg routing strategies*, IEEE/ACM Trans. Networking, 5 (1997), pp. 161–173.

[27] Y. A. Korilis, A. A. Lazar, and A. Orda, *Capacity allocation under noncooperative routing*, IEEE Trans. Automat. Control, 42 (1997), pp. 309–325.

[28] Y. A. Korilis, A. A. Lazar, and A. Orda, *Avoiding the Braess paradox in noncooperative networks*, J. Appl. Probab., 36 (1999), pp. 211–222.

[29] E. Koutsoupias, M. Mavronicolas, and P. Spirakis, *Approximate equilibria and ball fusion*, in Proceedings of the 9th Annual International Colloquium on Structural Information and Communication Complexity, Carleton Scientific, Waterloo, Canada, 2002, pp. 223–235.

[30] E. Koutsoupias and C. Papadimitriou, *Worst-case equilibria*, in Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, Berlin, 1999, pp. 404–413.

[31] D. C. Kozen, *Design and Analysis of Algorithms*, Springer-Verlag, New York, 1992.
[32] V. S. A. Kumar and M. V. Marathe, *Improved results for Stackelberg scheduling strategies*, in 29th International Colloquium on Automata, Languages, and Programming, Springer-Verlag, Heidelberg, 2002, pp. 776–787.
[33] A. Mas-Colell, M. D. Whinston, and J. R. Green, *Microeconomic Theory*, Oxford University Press, New York, 1995.
[34] M. Mavronicolas and P. Spirakis, *The price of selfish routing*, in Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing, ACM Press, New York, 2001, pp. 510–519.
[35] N. Nisan, *Algorithms for selfish agents: Mechanism design for distributed computation*, in Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, Berlin, 1999, pp. 1–15.
[36] N. Nisan and A. Ronen, *Algorithmic mechanism design*, Games Econ. Behav., 35 (2001), pp. 166–196.
[37] A. Orda, R. Rom, and N. Shimkin, *Competitive routing in multi-user communication networks*, IEEE/ACM Trans. Networking, 1 (1993), pp. 510–521.
[38] G. Owen, *Game Theory*, 3rd ed., Academic Press, New York, 1995.
[39] A. L. Peressini, F. E. Sullivan, and J. J. Uhl, *The Mathematics of Nonlinear Programming*, Springer-Verlag, New York, 1988.
[40] C. ReVelle and D. Serra, *The maximum capture problem including relocation*, INFOR, 29 (1991), pp. 130–138.
[41] A. Ronen, *Solving Optimization Problems among Selfish Agents*, Ph.D. thesis, Hebrew University of Jerusalem, 2000.
[42] T. Roughgarden, *Designing networks for selfish users is hard*, in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 472–481.
[43] T. Roughgarden, *The price of anarchy is independent of the network topology*, J. Comput. System Sci., 67 (2003), pp. 341–364.
[44] T. Roughgarden, *Selfish Routing*, Ph.D. thesis, Cornell University, Ithaca, NY, 2002.
[45] T. Roughgarden and É. Tardos, *How bad is selfish routing?*, J. ACM, 49 (2002), pp. 236–259.
[46] A. S. Schulz and N. S. Moses, *On the performance of user equilibria in traffic networks*, in Proceedings of the 14th Annual Symposium on Discrete Algorithms, ACM Press, New York, 2003, pp. 86–87.
[47] Y. Sheffi, *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1985.
[48] S. J. Shenker, *Making greed work in networks: A game-theoretic analysis of switch service disciplines*, IEEE/ACM Trans. Networking, 3 (1995), pp. 819–831.
[49] H. von Stackelberg, *Marktform und Gleichgewicht*, Springer-Verlag, Wein, 1934; English translation, *The Theory of the Market Economy*, published in 1952 by Oxford University Press.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 351–378

HYPERGRAPHS IN MODEL CHECKING: ACYCLICITY AND HYPERTREE-WIDTH VERSUS CLIQUE-WIDTH*

GEORG GOTTLOB† AND REINHARD PICHLER‡

Abstract. The principal aim of model checking is to provide efficient decision procedures for the evaluation of certain logical formulae over finite relational structures. Graphs and hypergraphs are important examples of such structures. If no restrictions are imposed on the logical formulae and on the structures under consideration, then this problem of model checking has a very high computational complexity. Hence, several restrictions have been proposed in the literature on the logical formulae and/or on the structures under consideration in order to guarantee the tractability of this decision problem, e.g., acyclicity, bounded tree-width, query-width and hypertree-width in the case of queries, as well as bounded tree-width and clique-width in the case of structures. The aim of this paper is a detailed comparison of the expressive power of these restrictions.

Key words. model checking, hypergraphs, tractability, database queries, clique-width, acyclicity, hypertree-width

AMS subject classifications. 03C13, 05C75, 68Q17, 68R10

DOI. 10.1137/S0097539701396807

1. Introduction.

Model checking is the problem of deciding whether a logical formula or query Q is satisfied by a finite structure $\mathcal{S}$, which is formally written as $\mathcal{S} \models Q$. Q may be a formula in first-order logic, monadic second-order logic, existential second-order logic, and so on. Model checking is a central issue in database systems [1], where $\mathcal{S}$ represents a database and the formula Q represents a database query. If Q is a closed formula, then Q is a Boolean query; otherwise $Q(\mathbf{x})$ with free variables $\mathbf{x}$ represents the query whose output consists of all tuples of domain values $\mathbf{a}$ such that $\mathcal{S} \models Q(\mathbf{a})$. Model checking is also a basic issue in the area of constraint satisfaction, which is essentially the same problem as conjunctive query evaluation [5, 33]. Finally, model checking is used in computer-aided verification [13], where $\mathcal{S}$ represents a state transition graph and Q is typically a formula of modal logic describing some temporal system behavior. The results of the present paper are, however, more relevant to the former applications, namely, conjunctive database queries and constraint satisfaction.

Without any further restriction on the form of the structures and/or the queries, these problems have a very high computational complexity. Hence, several restrictions have been proposed in the literature both for the structures and the queries in order to make these problems tractable. In particular, the evaluation problem for acyclic queries or for queries whose tree-width, query-width, or hypertree-width is bounded by some fixed constant k is tractable on arbitrary finite structures (combined complexity). On the other hand, arbitrary but fixed formulae of monadic second-order logic (precisely, so-called MS_1 formulae) can be evaluated in polynomial time on graphs whose tree-width or clique-width is bounded by some fixed constant k [16, 18, 19].

*Received by the editors October 19, 2001; accepted for publication (in revised form) September 29, 2003; published electronically February 18, 2004. This work was supported by the Austrian Science Fund (FWF), Project Z29-N04. A short version of this paper was presented at ICALP'01 (cf. [30]).

http://www.siam.org/journals/sicomp/33-2/39680.html

†Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9, A-1040 Vienna, Austria (gottlob@dbai.tuwien.ac.at).

‡Institut für Computersprachen, Technische Universität Wien, Favoritenstraße 9, A-1040 Vienna, Austria (reini@logic.at).

In other words, MS_1 queries have polynomial data complexity in the case of bounded tree-width or bounded query-width. MS_1 extends first-order logic by the possibility of quantifying over monadic relational variables representing sets of vertices. Note that only the concept of bounded tree-width has so far been applied both to the queries and the structures. On the other hand, acyclicity, bounded query-width, and hypertree-width have primarily been investigated as restrictions on the queries, while bounded clique-width has been considered only as a restriction on the structures. In this paper, we apply all of these restrictions both to the queries and to the structures. For reasons to be explained in section 2, we consider the clique-width of a hypergraph as the clique-width of its *incidence* graph (for definitions, see section 2.1). We shall thus answer the following questions:

(i) *Question* 1: How do the various notions of acyclicity and of bounded hypertree-width relate to the concept of bounded clique-width?

(ii) *Question* 2: Are Boolean conjunctive queries tractable if their clique-width is bounded by some fixed constant k?

(iii) *Question* 3: Bounded clique-width is currently the most general restriction on structures which makes the model checking problem for MS_1 formulae tractable. Can the tractability barrier be pushed any further by using known generalizations of acyclicity that are more powerful than clique-width?

As for the first question, we provide an exact classification of the expressive power of the various restrictions. The result is depicted in Figure 1.1. (Definitions of all these concepts are provided in section 2.1.) The arrows in the figure point from the less powerful concept to the more powerful one. In particular, it is shown in this paper that if a class $\mathcal{C}$ of hypergraphs is of bounded clique-width, then $\mathcal{C}$ is of bounded query-width (and, hence, also of bounded hypertree-width). Moreover, it is also shown, e.g., that β-acyclicity is uncomparable with bounded clique-width.

In [29] it was shown that the evaluation of a class $\mathcal{C}$ of Boolean conjunctive queries is tractable (actually, it is even in LOGCFL) if $\mathcal{C}$ is of bounded hypertree-width. Putting this together with our new result that bounded clique-width implies bounded query-width (see Figure 1.1), we immediately obtain that the evaluation of a class $\mathcal{C}$ of Boolean conjunctive queries is tractable if $\mathcal{C}$ is of bounded clique-width. Thus Question 2 is positively answered.

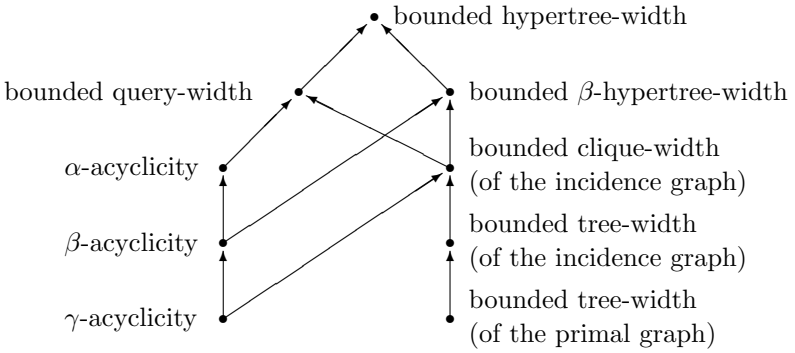


FIG. 1.1. *Expressive power of various restrictions on hypergraphs.*

As for the third question, we prove that the restriction to hypergraphs of bounded query-width or bounded β-hypertree-width is not sufficient to guarantee tractability of MS_1 queries. Thus bounded clique-width remains so far the most general restriction

on structures that guarantees the tractability of arbitrary fixed MS_1 queries.

While tree-width can be recognized in linear time [7], it is currently unclear whether bounded clique-width can be recognized in polynomial time. We therefore propose *generalized tree-width (gtw)*, a cyclicity measure located between tree-width and clique-width. It will be easy to see that bounded *gtw* is recognizable in polynomial time. Moreover, we shall prove that the evaluation of MS_1 queries over structures of bounded *gtw* is indeed tractable.

This paper is structured as follows: In section 2 we recall some basic notions and results. Restrictions on the form of the queries and of the structures will be considered in sections 3 and 4, respectively. In section 5, we show how the considerations of section 4 can be used for defining generalized tree-width. Finally, in section 6, we give some concluding remarks.

2. Preliminaries.

2.1. Various notions of width and acyclicity. A *graph* is a pair $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ consisting of a set $\mathcal{V}$ of vertices (or nodes) and a set $\mathcal{E}$ of edges. We consider only undirected graphs without self-loops and without multiple edges here. Hence, in particular, every edge $E \in \mathcal{E}$ is a two-element subset of $\mathcal{V}$. Two vertices are called *adjacent* iff they are the endpoints of an edge in $\mathcal{E}$. A set $\mathcal{W} \subseteq \mathcal{V}$ is a *module* of a graph iff the elements in $\mathcal{W}$ cannot be distinguished by the other vertices; i.e., every vertex in $\mathcal{V} - \mathcal{W}$ is adjacent either to all vertices $W \in \mathcal{W}$ or to none. A *subgraph* is a graph $\langle \mathcal{V}', \mathcal{E}' \rangle$ such that (s.t.) $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$ hold. A subgraph is *induced* iff $\mathcal{E}'$ is the restriction of $\mathcal{E}$ to those edges with both endpoints in $\mathcal{V}'$. In a *labeled graph*, every vertex has exactly one label. A *k-graph* is a labeled graph with k labels. Usually, these labels are taken from the set $\{1, \ldots, k\}$. A k-graph can be represented as $\langle \mathcal{V}, \mathcal{E}, \mathcal{V}_1, \ldots, \mathcal{V}_k \rangle$, where $\mathcal{V}$ is a set of vertices, $\mathcal{E}$ is a set of edges, and the sets $\mathcal{V}_1, \ldots, \mathcal{V}_k$ are (possibly empty) subsets of $\mathcal{V}$ that form a partition of $\mathcal{V}$.

A *hypergraph* is a pair $\mathcal{H} = \langle \mathcal{V}, \mathcal{E} \rangle$ consisting of a set $\mathcal{V}$ of vertices and a set $\mathcal{E}$ of hyperedges. A hyperedge $H \in \mathcal{E}$ is a subset of $\mathcal{V}$. A *subhypergraph* $\langle \mathcal{V}', \mathcal{E}' \rangle$ of $\langle \mathcal{V}, \mathcal{E} \rangle$ is obtained by deleting vertices and/or hyperedges; i.e., $\mathcal{V}' \subseteq \mathcal{V}$ and there exists a subset $\mathcal{F} \subseteq \mathcal{E}$ s.t. $\mathcal{E}' = \{H \cap \mathcal{V}' \mid H \in \mathcal{F}\}$. With every hypergraph $\mathcal{H} = \langle \mathcal{V}, \mathcal{E} \rangle$, we can associate the following two graphs: The *primal graph* (which is also called the *Gaifmann graph*) $\mathcal{P}(\mathcal{H})$ has the same vertices $\mathcal{V}$ as $\mathcal{H}$. Moreover, two vertices $V_1, V_2 \in \mathcal{V}$ are connected by an edge in $\mathcal{P}(\mathcal{H})$ iff there is a hyperedge $H \in \mathcal{E}$ s.t. both V_1 and V_2 are contained in $\mathcal{E}$. The *incidence graph* $\mathcal{I}(\mathcal{H})$ is a bipartite graph with vertices in $\mathcal{V} \cup \mathcal{E}$. Moreover, there is an edge in $\mathcal{I}(\mathcal{H})$ between two vertices $V \in \mathcal{V}$ and $H \in \mathcal{E}$ iff (in the hypergraph $\mathcal{H}$) V occurs in the hyperedge H. The incidence graph can be considered either as an unlabeled graph or as a 2-graph, with the labels $\mathcal{V}$ and $\mathcal{E}$, respectively.

In order to determine the *clique-width* of a (labeled or unlabeled) graph, we have to deal with so-called *k-expressions* t and the graphs $G(t)$ generated by such k-expressions:

(i) Let $i \in \{1, \ldots, k\}$ and let V be a vertex. Then $i(V)$ is a k-expression. The corresponding graph consists of a single vertex V whose label is i.

(ii) Let r and s be k-expressions that have no vertices in common. Then $r \oplus s$ is also a k-expression. The graph thus generated is the (disjoint) union of the graphs $G(r)$ and $G(s)$.

(iii) Let $i, j \in \{1, \ldots, k\}$ with $i \neq j$ and let r be a k-expression. Then $\eta_{i,j}(r)$ is also a k-expression. The corresponding graph is the same as $G(r)$ augmented by edges from every vertex with label i to every vertex with label j.

(iv) Let $i, j \in \{1, \ldots, k\}$ with $i \neq j$ and let r be a k-expression. Then $\rho_{i \to j}(r)$ is also a k-expression whose graph is the same as $G(r)$ except that all vertices with label i in $G(r)$ are relabeled to j.

The graph generated by a k-expression t can be considered either as a labeled graph with labels in $\{1, \ldots, k\}$ or as an unlabeled graph (by ignoring the labels assigned by t). Every subexpression s of a k-expression t generates a subgraph $G(s)$ of $G(t)$ (when considered as an unlabeled graph). The clique-width $cw(\mathcal{G})$ of a (labeled or unlabeled) graph $\mathcal{G}$ is the minimum k s.t. there exists a k-expression that generates $\mathcal{G}$. Obviously, $cw(\mathcal{G}) \leq n$ for every graph with n vertices. It can be shown that every clique has clique-width 2; e.g., the clique with four nodes V_1, V_2, V_3, V_4 can be generated by the 2-expression $\eta_{1,2}(2(V_4) \oplus \rho_{2 \to 1}(\eta_{1,2}(2(V_3) \oplus \rho_{2 \to 1}(\eta_{1,2}(2(V_2) \oplus 1(V_1))))))$.

A *tree decomposition* of a graph $\langle \mathcal{V}, \mathcal{E} \rangle$ is a pair $\langle T, \lambda \rangle$, where $T = \langle N, F \rangle$ is a tree and λ is a labeling function with $\lambda(p) \subseteq \mathcal{V}$ for every node $p \in N$ s.t. the following conditions hold:

(i) $\forall V \in \mathcal{V}$, $\exists p \in N$ s.t. $V \in \lambda(p)$.

(ii) $\forall E \in \mathcal{E}$ with endpoints V_1 and V_2, $\exists p \in N$ s.t. $V_1 \in \lambda(p)$ and $V_2 \in \lambda(p)$.

(iii) $\forall V \in \mathcal{V}$, the set $\{p \in N : V \in \lambda(p)\}$ induces a connected subtree of T.

The width of a tree decomposition $\langle T, \lambda \rangle$ is $\max(\{|\lambda(p)| - 1 : p \in N\})$. The tree-width $tw(\mathcal{G})$ of a graph $\mathcal{G}$ is the minimum width over all its tree decompositions.

A *join tree* of a hypergraph $\langle \mathcal{V}, \mathcal{H} \rangle$ is a labeled tree $\langle T, \lambda \rangle$ with $T = \langle N, F \rangle$ and a labeling function λ with $\lambda(p) \in \mathcal{H}$ for every node $p \in N$. Moreover, the following conditions hold:

(i) $\forall H \in \mathcal{H}$, $\exists p \in N$ s.t. $\lambda(p) = H$.

(ii) "Connectedness condition": Let $\lambda(p_1) = H_1$ and $\lambda(p_2) = H_2$ for two distinct nodes p_1 and p_2. Moreover, suppose that some vertex $V \in \mathcal{V}$ occurs in both hyperedges H_1 and H_2. Then V must also occur in all hyperedges that are used as labels on the path from p_1 to p_2.

A hypergraph is *α-acyclic* iff it has a join tree.

In [12], a *query decomposition* of a hypergraph $\langle \mathcal{V}, \mathcal{H} \rangle$ is defined as a pair $\langle T, \lambda \rangle$ where $T = \langle N, F \rangle$ is a tree and λ is a labeling function with $\lambda(p) \subseteq (\mathcal{V} \cup \mathcal{H})$ for every $p \in N$ and

(i) $\forall H \in \mathcal{H}\ \exists p \in N$ s.t. $H \subseteq \{V \mid V \in \lambda(p) \cap \mathcal{V}$ or $\exists H' \in \lambda(p) \cap \mathcal{H}$ with $V \in H'\}$;

(ii) "Connectedness condition": $\forall V \in \mathcal{V}$, the set $\{p \in N : V \in \lambda(p)\} \cup \{q \in N : \exists H \in \mathcal{H}$ s.t. $H \in \lambda(q)$ and V occurs in the hyperedge $H\}$ induces a connected subtree of T.[1]

The width of a query decomposition $\langle T, \lambda \rangle$ is $\max(\{|\lambda(p)| : p \in N\})$. The query-width $qw(\mathcal{H})$ of a hypergraph $\mathcal{H}$ is the minimum width over all its query decompositions. From [12] we know that a hypergraph $\mathcal{H}$ is α-acyclic iff $qw(\mathcal{H}) = 1$ holds.

In [29], a *hypertree decomposition* of a hypergraph $\langle \mathcal{V}, \mathcal{H} \rangle$ is defined as a triple $\langle T, \chi, \lambda \rangle$ where $T = \langle N, F \rangle$ is a tree and χ and λ are labeling functions with $\chi(p) \subseteq \mathcal{V}$ and $\lambda(p) \subseteq \mathcal{H}$ for every $p \in N$. Moreover, the following conditions hold:

(i) $\forall H \in \mathcal{H}$, $\exists p \in N$ s.t. $H \subseteq \chi(p)$ i.e., "p covers H."

(ii) "Connectedness condition": $\forall V \in \mathcal{V}$, the set $\{p \in N : V \in \chi(p)\}$ induces a connected subtree of T.

(iii) $\forall p \in N$, $\chi(p)$ contains only vertices that actually occur in at least one hyperedge of $\lambda(p)$.

[1] Note that in the original definition in [12], it is also required that $\forall H \in \mathcal{H}$, the set $\{p \in N : H \in \lambda(p)\}$ is a connected subtree of T. However, this restriction is of no use as far as the tractability of the evaluation of queries is concerned. We have therefore omitted this condition here.

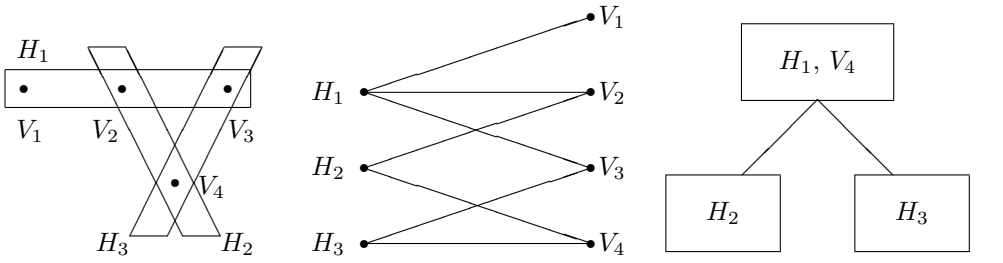
(iv) For every $p \in N$, if a vertex V occurs in some hyperedge $H \in \lambda(p)$ and if V is contained in $\chi(q)$ for some node q in the subtree below p, then V must also be contained in $\chi(p)$.

The width of a hypertree decomposition $\langle T, \chi, \lambda \rangle$ is $\max(\{|\lambda(p)| : p \in N\})$. The hypertree-width $hw(\mathcal{H})$ of a hypergraph $\mathcal{H}$ is the minimum width over all its hypertree decompositions.

A conjunctive query Q is a first-order formula in prenex form whose only connectives are $\exists$ and $\wedge$. With every conjunctive query, we can associate a hypergraph $\mathcal{H}$, whose vertices $V_1, \ldots, V_n$ correspond to the variables $x_1, \ldots, x_n$ occurring in Q. Moreover, for every atom A with variables $Var(A) = \{x_{i_1}, \ldots, x_{i_\alpha}\}$, there is a hyperedge $H = \{V_{i_1}, \ldots, V_{i_\alpha}\}$ in the hypergraph, and vice versa. Then the notions of hypertree-width, query-width, and acyclicity carry over in a natural way from hypergraphs to conjunctive queries. Likewise, the incidence graph or the primal graph of a conjunctive query Q is simply the corresponding graph of the associated hypergraph $\mathcal{H}$. Actually, the clique-width or tree-width of a hypergraph $\mathcal{H}$ can be defined as the corresponding width of either the incidence graph or the primal graph. If not indicated otherwise, we shall assume that the clique-width and the tree-width of a hypergraph $\mathcal{H}$ refer to the incidence graph (considered as an unlabeled graph). As a justification of this choice, note that there are NP-hard classes of queries s.t. the clique-width of their primal graphs is bounded by some fixed constant k. Consider, for example, the class $\mathcal{C}$ of conjunctive queries $CLIQUE_m$ asking whether a graph $(\mathcal{V}, \mathcal{E})$ has a clique of size m for integers m. This class of queries is well known to be NP-hard (cf. [26, problem GT19]). $CLIQUE_m$ is expressed by the formula $\exists x_1, \exists x_2, \ldots, \exists x_m : \bigwedge_{1 \leq i < j \leq m} ((x_i \neq x_j) \wedge E(x_i, x_j))$. The primal graph associated to $CLIQUE_m$ is a clique of m vertices and has clique-width $k = 2$ for any $m > 1$. Thus, *Question* 2 asked in the introduction has a trivial negative answer in case the clique-width of a query is defined based on its primal graph. The question becomes highly nontrivial if instead, as done in the present paper, we define the clique-width of a query (or of a hypergraph) on the basis of its incidence graph.

Clique-width and tree-width are *hereditary* properties in that $cw(\mathcal{G}') \leq cw(\mathcal{G})$ and $tw(\mathcal{G}') \leq tw(\mathcal{G})$ hold for every induced subgraph $\mathcal{G}'$ of a graph $\mathcal{G}$. Moreover, any subhypergraph $\mathcal{H}'$ of a hypergraph $\mathcal{H}$ gives rise to an induced subgraph $\mathcal{I}(\mathcal{H}')$ of the incidence graph $\mathcal{I}(\mathcal{H})$. Hence, if $\mathcal{H}'$ is an arbitrary subhypergraph of $\mathcal{H}$, then $cw(\mathcal{H}') \leq cw(\mathcal{H})$ and $tw(\mathcal{H}') \leq tw(\mathcal{H})$ hold, where cw and tw are defined via the incidence graph of a hypergraph. In contrast, α-acyclicity, query-width, and hypertree-width do not share this property, e.g., a hypergraph $\mathcal{H}$ can be α-acyclic even though some subhypergraph $\mathcal{H}'$ is not. Likewise, $\mathcal{H}$ can have a subhypergraph $\mathcal{H}'$ s.t. $qw(\mathcal{H}) < qw(\mathcal{H}')$ or $hw(\mathcal{H}) < hw(\mathcal{H}')$ hold. The notions of *β-acyclicity* and *β-hypertree-width* can be regarded as the hereditary counterparts of α-acyclicity and hypertree-width: In [24], a hypergraph $\mathcal{H}$ is defined to be β-acyclic iff every (not necessarily induced) subhypergraph $\mathcal{H}'$ of $\mathcal{H}$ is α-acyclic. Analogously, we can define the β-hypertree-width of $\mathcal{H}$ as the $\max(\{hw(\mathcal{H}') : \mathcal{H}' \text{ is a subhypergraph of } \mathcal{H}\})$. In [24], another notion of acyclicity is presented, namely, γ-acyclicity. Any hypergraph that is γ-acyclic is also β-acyclic. An algorithmic definition of γ-acyclicity will be given in section 4.1.

In this work, we shall compare the expressive power of the above defined notions of width. Let $x, y \in \{\text{tree, clique, query, hypertree, } \beta\text{-hypertree}\}$. Moreover, let $\mathcal{C}$ be a class of hypergraphs (or graphs). We say that $\mathcal{C}$ is of *bounded x-width* if there exists some constant k s.t. every hypergraph (or graph, respectively) in $\mathcal{C}$ has x-width less than or equal to k. Moreover, we say that *bounded x-width implies bounded y-width* if



(a) hypergraph $\mathcal{H}$ (b) incidence graph $\mathcal{I}(\mathcal{H})$ (c) query decomposition QD

FIG. 2.1. *$\mathcal{H}$, $\mathcal{I}(\mathcal{H})$, and QD.*

every class $\mathcal{C}$ of hypergraphs (or graphs) that is of bounded x-width is also of bounded y-width.

We conclude this section with an example, which should help to illustrate some of the main concepts used in this paper.

Example. Consider the *conjunctive query* $Q = A(x_1, x_2, x_3) \wedge B(x_2, x_4) \wedge C(x_3, x_4)$ consisting of 3 atoms and 4 variables. Consequently, the corresponding *hypergraph* $\mathcal{H} = \langle \mathcal{V}, \mathcal{E} \rangle$ is made up of 4 vertices $\mathcal{V} = \{V_1, V_2, V_3, V_4\}$ and 3 hyperedges $\mathcal{E} = \{H_1, H_2, H_3\}$ with $H_1 = \{V_1, V_2, V_3\}$, $H_2 = \{V_2, V_4\}$, and $H_3 = \{V_3, V_4\}$. Then the *incidence graph* $\mathcal{I}(\mathcal{H})$ is the bipartite graph with vertices in $\{V_1, V_2, V_3, V_4, H_1, H_2, H_3\}$ s.t. 2 nodes V_i and H_j are adjacent in $\mathcal{I}(\mathcal{H})$ iff, in the hypergraph $\mathcal{H}$, V_i occurs in the hyperedge H_j.

The hypergraph $\mathcal{H}$ and the incidence graph $\mathcal{I}(\mathcal{H})$ are displayed in Figure 2.1. Moreover, a *query decomposition* QD of $\mathcal{H}$ is shown there also. Note that QD has width 2. Due to the cycle $\{V_2, V_3\}$, $\{V_3, V_4\}$, $\{V_4, V_2\}$, the hypergraph $\mathcal{H}$ is not acyclic and, therefore, $\mathcal{H}$ cannot have a query decomposition of width 1. Hence, we have in fact $qw(\mathcal{H}) = 2$. Finally, a k-expression t (with $k = 5$) generating the graph $\mathcal{I}(\mathcal{H})$ can be obtained via the following algorithm:

Initialization (nodes H_i): First we introduce all nodes H_1, H_2, and H_3 with pairwise distinct labels; i.e., we set $s_0 := 1(H_1) \oplus 2(H_2) \oplus 3(H_3)$.

Iteration (nodes V_j): In a loop over all nodes $V_1, \ldots, V_4$, we carry out the following steps: Introduce the node V_j with label 4, draw all required edges between V_j and the H_i's and relabel V_j to 5, i.e.,

$$s_1 := \rho_{4\to5}(\eta_{4,1}(4(V_1) \oplus s_0)), \qquad s_2 := \rho_{4\to5}(\eta_{4,1}(\eta_{4,2}(4(V_2) \oplus s_1))),$$

$$s_3 := \rho_{4\to5}(\eta_{4,1}(\eta_{4,3}(4(V_3) \oplus s_2))), \quad s_4 := \rho_{4\to5}(\eta_{4,2}(\eta_{4,3}(4(V_4) \oplus s_3))).$$

Then s_4 is the desired k-expression that generates $\mathcal{I}(\mathcal{H})$. Note that the above algorithm is applicable to any bipartite graph. Hence, in any bipartite graph $\mathcal{B}$ with nodes in $\mathcal{N}_1 \cup \mathcal{N}_2$, the condition $cw(\mathcal{B}) \leq \min(|\mathcal{N}_1|, |\mathcal{N}_2|) + 2$ is fulfilled.

2.2. Tractability via bounded width or acyclicity. In model checking, one is interested in the (efficient) evaluation of certain logical formulae (= "queries") over finite relational structures. To this end, the various notions of width and acyclicity recalled in the previous section have been explored in two principal ways: They have been used to restrict either the relational structures or the queries. Restrictions on the

structures in terms of tree-width and clique-width have been investigated in the area of graph grammars and graph algorithms. On the other hand, restrictions imposed on the queries such as the various forms of acyclicity as well as bounded query-width and hypertree-width have been mainly analyzed in database theory and constraint satisfaction. Note that, so far, only tree-width is common ground for both directions of research.

In this paper, we shall first deal with restrictions on the form of the queries. Recall that the evaluation of arbitrary first-order queries is PSPACE-complete (cf. [34, 35]). Actually, even if we restrict the form of first-order queries to conjunctive queries (where only conjunctions and existential quantification are allowed), then the query evaluation is still NP-complete (see [11]). If conjunctive queries are further restricted to α-acyclic conjunctive queries, then this problem becomes tractable (cf. [36]). However, acyclicity is a very severe restriction. Hence, in recent years, several attempts to deal with "almost acyclic queries" have been made. In particular, several notions of width have been introduced in order to extend the class of tractable conjunctive queries, namely, tree-width, query-width, and hypertree-width (cf. [12, 25, 29, 31, 33]). In [28] and [29], it has been shown that, for some fixed k, the class of conjunctive queries with hypertree-width $\leq k$ properly contains the classes where the tree-width (of the incidence graph or of the primal graph) or the query-width, respectively, is bounded by k. Moreover, the concept of hypertree-width is a generalization of α-acyclicity in that a conjunctive query is acyclic iff it has hypertree-width 1.

In [15], the complexity of testing certain graph properties is investigated. In terms of model checking, this corresponds to evaluating a fixed query over finite graphs. If the queries are (arbitrary but fixed) first-order formulae, then this problem is tractable for all finite graphs without any further restrictions. However, the expressive power of first-order logic is comparatively weak. Hence, attempts were made to investigate larger classes of queries. In fact, many interesting graph properties like 3-colorability, Hamiltonian circuit, partition into triangle, etc. (cf. [15, 26]), are expressible as *monadic second-order* queries. Note that there are basically two ways of representing a graph by a logical structure, namely, either the domain consists of both vertices and edges or the domain consists of vertices only. In the former case, quantified variables of a monadic second-order formula may refer to edges or vertices, whereas in the latter case, only quantification over vertices is allowed. Formulae in the former case are referred to as MS_1 formulae, while formulae in the latter case are called MS_2 formulae. It has been shown that MS_2 formulae can be evaluated in polynomial time (in fact, even linear time suffices) over a class $\mathcal{C}$ of graphs if $\mathcal{C}$ is of bounded tree-width. The restriction to bounded clique-width has proved to allow for a much larger class of structures than bounded tree-width. In particular, bounded tree-width implies bounded clique-width (cf. [19]), while the converse is in general not true; e.g., the class of cliques is of bounded clique-width (where the bound is simply 2) but of unbounded tree-width. It has been shown that the evaluation of fixed MS_1 formulae over a class $\mathcal{C}$ of graphs is tractable if $\mathcal{C}$ is of bounded clique-width (cf. [15, 16, 17]).

3. Restricting the form of the queries. In this section we consider the case of conjunctive queries over arbitrary relational structures, where the queries are subjected to restrictions that guarantee tractability. It has already been noted in the previous section that bounded hypertree-width is the most powerful concept studied so far. We shall now show that bounded clique-width does not allow for a bigger class of conjunctive queries than does bounded hypertree-width. More precisely, in the

proof of Theorem 3.1, we shall provide an algorithm that, when given a k-expression for (the incidence graph of) some hypergraph $\mathcal{H}$, constructs a query decomposition of $\mathcal{H}$ whose width is $\leq k$.

It is convenient to introduce some additional notation first. In the proof of Theorem 3.1, we are going to deal with three kinds of graphs or hypergraphs, respectively, i.e., a hypergraph $\mathcal{H}'$, the incidence graph $\mathcal{I}'$ of $\mathcal{H}'$, and the query decomposition QD' of $\mathcal{H}'$. We shall therefore speak about H-vertices, I-vertices, and Q-vertices (or, equivalently, H-nodes, I-nodes, and Q-nodes) when referring to the vertices in $\mathcal{H}'$, $\mathcal{I}'$, or QD', respectively. Likewise, we shall encounter two kinds of labels, namely, the labels assigned by the k-expression t' and the labels of the Q-nodes in the query decomposition QD'. In order to avoid confusion, we shall refer to these labels as t-labels (for the labels of the I-vertices according to the k-expression t') and Q-labels (in case of the Q-nodes), respectively. Strictly speaking, t' assigns t-labels to the nodes in $\mathcal{I}'$ (i.e., the I-nodes). However, every I-node uniquely corresponds either to a hyperedge or to an H-vertex in $\mathcal{H}'$. Hence, as an abbreviation, we shall speak about the "t-label of a hyperedge" or the "t-label of an H-vertex" when we refer to the t-label of the I-vertex corresponding to this hyperedge or H-vertex, respectively. We shall say that a Q-node q of the query decomposition QD' "*covers*" an H-vertex V iff either V is contained in the Q-label $\lambda'(q)$ or $\lambda'(q)$ contains some hyperedge H s.t. V is an H-vertex occurring in this hyperedge H. Finally, for every $\ell \in \{1, \ldots, k\}$, we define the set $\mathcal{Q}_\ell(QD')$ of Q-nodes as $\mathcal{Q}_\ell(QD') = \{p : p \text{ is a } Q\text{-node in } QD' \text{ and the } Q\text{-label of } p \text{ contains a hyperedge or an } H\text{-vertex with } t\text{-label } \ell\}$.

THEOREM 3.1 (query-width is bounded by clique-width). *Let $\mathcal{H}$ be an arbitrary hypergraph with incidence graph $\mathcal{I}(\mathcal{H})$. Then $qw(\mathcal{H}) \leq cw(\mathcal{I}(\mathcal{H}))$ holds.*

Proof. Let $\mathcal{H}$ be a hypergraph with incidence graph $\mathcal{I}$ and let t be a k-expression that generates $\mathcal{I}$. Note that every subexpression t' of t generates a subgraph $\mathcal{I}'$ of the incidence graph $\mathcal{I}$. Moreover, every such $\mathcal{I}'$ uniquely defines a hypergraph $\mathcal{H}'$. Then we construct a query decomposition QD' of $\mathcal{H}'$ inductively on the structure of t' with the following properties:

1. *QD' is a query decomposition of $\mathcal{H}'$ of width $\leq k$.*
2. *For every $\ell \in \{1, \ldots, k\}$, the above defined set $\mathcal{Q}_\ell(QD')$ of Q-vertices, if not empty, forms a connected subtree of QD' s.t. the root of this subtree coincides with the root of QD' itself. Moreover, if the incidence graph $\mathcal{I}'$ actually contains an I-node with t-label ℓ, then the Q-label $\lambda(r)$ of the root r of QD' also contains at least one I-node with t-label ℓ.*
3. *Suppose that p is a Q-node in QD', V is an H-vertex with t-label ℓ, and p covers V. Moreover, let the Q-node q be the parent of p in QD'. Then either q also covers V or the Q-label $\lambda'(q)$ contains some I-node N whose t-label is ℓ.*

For the construction of QD', we consider each of the four basic operations of a k-expression separately.

Introduction of a new vertex. Let $t' = i(N)$ for some node N in $\mathcal{I}$; i.e., N either corresponds to a hyperedge or to an H-vertex in $\mathcal{H}'$. In either case, the corresponding query decomposition QD' consists of a single node r whose Q-label is the singleton $\{N\}$. Thus, QD' trivially fulfills the above conditions 1 through 3.

Disjoint union. Let $t' = s_1 \oplus s_2$. Moreover, let $\mathcal{J}_1$ and $\mathcal{J}_2$ be the (disjoint) subgraphs of $\mathcal{I}$ defined by s_1 and s_2, respectively, and let $\mathcal{H}_1$ and $\mathcal{H}_2$ be the corresponding hypergraphs. By the induction hypothesis, there exist query decompositions $QD_1 = \langle T_1, \lambda_1 \rangle$ and $QD_2 = \langle T_2, \lambda_2 \rangle$ of $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively, for which the above three conditions hold. Let r_1 and r_2 denote the root nodes of T_1 and T_2, respectively.

Then we construct the new query decomposition $QD' = \langle T', \lambda' \rangle$ in the following way: T' has a new root node r and the subtrees T_1 and T_2 s.t. r_1 and r_2 are the child nodes of r. As for the labeling function λ', the Q-labels of the Q-nodes in QD_1 and QD_2 are left unchanged. The Q-label $\lambda'(r)$ of the new root r is defined from the Q-labels of r_1 and r_2 as follows: Let $\mathcal{R} = \lambda_1(r_1) \cup \lambda_2(r_2)$. By assumption, the H-vertices and hyperedges in $\mathcal{R}$ are assigned at most k different t-labels by t'. Then we construct the Q-label $\lambda'(r)$ by selecting one representative from $\mathcal{R}$ for each t-label according to t'. Of course, there can be at most k such representatives. We shall now show that conditions 1 through 3 hold for QD':

(i) *Condition* 1. By the above construction of λ' and by the induction hypothesis, the width of QD' is clearly $\leq k$. It remains to prove that QD' is indeed a query decomposition of $\mathcal{H}'$. By the induction hypothesis, every hyperedge of $\mathcal{H}_1$ occurs in some Q-label of QD_1 and every hyperedge of $\mathcal{H}_2$ occurs in some Q-label of QD_2 and, thus, also in some Q-label of QD'. Moreover, the connectedness condition follows from the induction hypothesis and the fact that $\mathcal{I}'$ is obtained as the *disjoint* union of $\mathcal{J}_1$ and $\mathcal{J}_2$. Hence, in particular, the hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ have no H-vertices in common.

(ii) *Condition* 2. Let $\ell \in \{1, \ldots, k\}$. By the induction hypothesis, $\mathcal{Q}_\ell(QD_1)$ and $\mathcal{Q}_\ell(QD_2)$ form connected subtrees of the query decompositions QD_1 and QD_2, respectively. Moreover, the roots of these subtrees coincide with the roots of QD_1 and QD_2, respectively, and if $\mathcal{J}_1$ or $\mathcal{J}_2$ contains an I-node with label ℓ, then $\lambda_1(r_1)$ or $\lambda_1(r_2)$, respectively, indeed contains an I-node with t-label ℓ. By our construction, $\mathcal{Q}_\ell(QD')$ is obtained as follows:

$$\mathcal{Q}_\ell(QD') = \begin{cases} \mathcal{Q}_\ell(QD_1) & \\ \quad \cup\ \mathcal{Q}_\ell(QD_2) \cup \{r\} & \text{if } s_1 \text{ and } s_2 \text{ contain an } I\text{-node with } t\text{-label } \ell, \\ \mathcal{Q}_\ell(QD_1) \cup \{r\} & \text{if only } s_1 \text{ contains an } I\text{-node with } t\text{-label } \ell, \\ \mathcal{Q}_\ell(QD_2) \cup \{r\} & \text{if only } s_2 \text{ contains an } I\text{-node with } t\text{-label } \ell, \\ \emptyset & \text{otherwise.} \end{cases}$$

In all of these four cases, condition 2 clearly holds.

(iii) *Condition* 3. Let p be a Q-node in QD' and let q be the parent of p in QD'. In particular, p is not the new root node r in QD' and, therefore, p already existed before, say, in QD_1. Now suppose that V is an H-vertex with t-label ℓ s.t. p covers V in QD'. If p is the root r_1 of QD_1, then q is the new root r in QD', which (by condition 2) contains some I-node N with t-label ℓ. On the other hand, if $p \neq r_1$, then also q is a Q-node of QD_1 and condition 3 holds by the induction hypothesis.

Introduction of edges. Let $t' = \eta_{i,j}(s_1)$. Moreover, let $\mathcal{J}_1$ be the subgraph of $\mathcal{I}$ defined by s_1 and let $\mathcal{H}_1$ be the corresponding hypergraph. Note that the $\eta_{i,j}$ operation may have added some new edges in the incidence graph and, therefore, a hyperedge H in $\mathcal{H}'$ will, in general, contain more H-vertices than if we consider H as a hyperedge in $\mathcal{H}_1$. In fact, we may assume without loss of generality (w.l.o.g.) that the application of $\eta_{i,j}$ to s_1 indeed creates a new edge in $\mathcal{I}'$, which did not exist in $\mathcal{J}_1$. Otherwise, the hypergraphs $\mathcal{H}'$ and $\mathcal{H}_1$ would be identical and the query decomposition QD_1 of $\mathcal{H}_1$ would also be the desired query decomposition of $\mathcal{H}'$.

In order to distinguish between hyperedges in $\mathcal{H}'$ and $\mathcal{H}_1$, we shall write H and H^-, respectively; i.e., by H^-, we denote a hyperedge in $\mathcal{H}_1$ and by H we denote the corresponding hyperedge in $\mathcal{H}'$. Of course, we have $H^- \subseteq H$ but, in general, $H^- = H$ is not true.

By the induction hypothesis, there exists a query decomposition $QD_1 = \langle T_1, \lambda_1 \rangle$ of $\mathcal{H}_1$ for which the above three conditions hold. Then we construct the query de-

composition $QD' = \langle T_1, \lambda' \rangle$ of $\mathcal{H}'$ in such a way that the tree T_1 is left unchanged. As for the labeling function λ' of QD', recall that $\mathcal{I}'$ is a bipartite graph. Moreover, by assumption, $\eta_{i,j}$ creates at least one new edge in $\mathcal{I}'$. Hence, we may assume w.l.o.g. that i is the t-label of H-vertices only and j is the t-label of hyperedges only. By condition 2 of the induction hypothesis, $\mathcal{Q}_j(QD_1)$ is a connected subtree of QD_1 whose root coincides with the root r_1 of QD_1. Thus, the Q-label $\lambda_1(r_1)$ contains a hyperedge G^- with t-label j. Then λ' is defined as follows: The Q-label of the root r_1 is left unchanged, i.e., $\lambda'(r_1) := \lambda_1(r_1)$. Likewise, if the Q-label of some Q-node $p \neq r_1$ does not contain an H-vertex with label i, then we do not alter this Q-label. In the Q-label of all other Q-nodes, we replace the H-vertices with t-label i by the hyperedge G.[2]

Now it can be easily checked that condition 2 holds for QD'; i.e., by our construction of the labeling function λ', the sets $\mathcal{Q}_\ell(QD')$ are obtained from $\mathcal{Q}_\ell(QD_1)$ for $\ell \in \{1, \ldots, k\}$ in the following way:

$$\mathcal{Q}_\ell(QD') = \begin{cases} \{r_1\} & \text{if } \ell = i, \\ \mathcal{Q}_i(QD_1) \cup \mathcal{Q}_j(QD_1) & \text{if } \ell = j, \\ \mathcal{Q}_\ell(QD_1) & \text{otherwise.} \end{cases}$$

In all of these three cases, condition 2 obviously holds.

For the proof of condition 3, let p be a Q-node in QD' and let q be the parent of p in QD'. Moreover, let V be an H-vertex with t-label ℓ s.t. p covers V. Then we have to show that either q also covers V or the Q-label $\lambda'(q)$ contains some I-node N whose t-label is ℓ. Of course, p and q are also Q-nodes in QD_1. Then we distinguish the following cases:

Case 1. Suppose that $\ell \neq i$. If V occurs in some hyperedge H of the Q-label $\lambda'(p)$ s.t. the corresponding hyperedge H^- already occurred in the Q-label $\lambda_1(p)$ in QD_1, then condition 3 follows immediately from the induction hypothesis. Likewise, if V itself is contained in $\lambda'(p)$, then V was already contained in $\lambda_1(p)$, and we may again conclude by the induction hypothesis that condition 3 still holds for QD'. On the other hand, suppose that V occurs in some hyperedge H from the Q-label $\lambda'(p)$ s.t. the corresponding hyperedge H^- did not occur in the Q-label $\lambda_1(p)$ in QD_1. In other words, $H = G$ and G was introduced into $\lambda'(p)$ when we replaced the H-vertices with label i by the hyperedge G in the Q-labels of all Q-nodes except for the root node r_1 of QD'. If $q = r_1$, then condition 3 clearly holds, since $\lambda'(q)$ also contains G. Otherwise, by condition 2 of the induction hypothesis, the parent node q of p in QD_1 also contains some I-node N with t-label i in its Q-label $\lambda_1(q)$. By our construction, N is replaced by the hyperedge G in $\lambda'(q)$. Hence, V is also covered by q in QD'.

Case 2. Suppose that $\ell = i$. W.l.o.g., we may assume that $q \neq r_1$, since otherwise q contains some I-node with t-label i by condition 2, and we are done. By our construction, V (with t-label i) itself cannot occur in the Q-label of the Q-node $p \neq r_1$ in QD'. Hence, there exists some hyperedge H in $\mathcal{H}'$ s.t. V occurs in H and $H \in \lambda'(p)$. We distinguish two subcases for the t-label of H:

(i) If H has the t-label j, then, by condition 2, $\lambda'(q)$ also contains some hyperedge H' with t-label j. Moreover, by the application of the $\eta_{i,j}$ operation in the

[2]Actually, this formulation is slightly inaccurate. Recall that, in general, a hyperedge H in $\mathcal{H}'$ may have some additional vertices, which are not contained in the corresponding hyperedge H^- in $\mathcal{H}_1$; e.g., strictly speaking, we would have to write $\lambda'(r_1) := \{H : H^- \in \lambda_1(r_1)\} \cup \{V \in \lambda_1(r_1)\}$ rather than $\lambda'(r_1) := \lambda_1(r_1)$. However, the meaning of the latter formulation is clear. Moreover, as far as the incidence graph is concerned, H^- and H refer to exactly the same I-node, anyway.

incidence graph $\mathcal{I}'$, the hyperedge H' in $\mathcal{H}'$ indeed contains all H-vertices with t-label i. Thus, q clearly covers V in QD'.

(ii) Finally, suppose that H has a t-label different from j. Then H^- already existed in $\lambda_1(p)$ and the H-vertex V already occurred in the hyperedge H^- of the hypergraph $\mathcal{H}_1$. Hence, we may apply condition 3 of the induction hypothesis; i.e., either V occurs in some hyperedge F^- in $\mathcal{H}_1$ s.t. $F^- \in \lambda_1(q)$ or $\lambda_1(q)$ contains some I-node N whose t-label is i. In the former case, $F \in \lambda'(q)$ and, therefore, q also covers V in QD'. In the latter case, N is replaced by G in $\lambda'(q)$ and, again, q covers V in QD'.

It remains to prove that also condition 1 still holds; i.e., QD' indeed is a query decomposition of $\mathcal{H}'$ with width $\leq k$. Actually, the bound on the width follows immediately from the induction hypothesis and the fact that $|\lambda'(p)| \leq |\lambda_1(p)|$ holds for every Q-node $p \in T_1$. Moreover, for every hyperedge H in $\mathcal{H}'$, QD_1 contains a Q-node p with $H^- \in \lambda_1(p)$ and, therefore, $H \in \lambda'(p)$ also holds. The only difficult part of the proof is to show that QD' fulfills the connectedness condition. In fact, this is the only place in the whole proof of Theorem 3.1 where we actually need conditions 2 and 3 of our construction.

Now let V be an arbitrary H-vertex of $\mathcal{H}'$ and let ℓ denote the t-label of V. Then we have to show that the set of Q-nodes that cover V form a connected subtree in QD'. To this end we distinguish the following cases:

Case 1. Let $\ell \neq i$. In this case, for every hyperedge H in $\mathcal{H}'$ with $V \in H$, we know that $V \in H^-$ also holds. Moreover, by the induction hypothesis, the set of Q-nodes $\mathcal{P}_1 = \{p \in T_1 : V \in \lambda_1(p)\} \cup \{q \in T_1 : \exists H^- \text{ s.t. } H^- \text{ is a hyperedge in } \mathcal{H}_1,\ H^- \in \lambda_1(q) \text{ and } V \in H^-\}$ induces a connected subtree of T_1. First, suppose that V does not occur in the hyperedge G by which all H-vertices with t-label i were replaced when we constructed λ' from λ_1. In this case, the set of Q-nodes $\mathcal{P}' = \{p \in T_1 : V \in \lambda'(p)\} \cup \{q \in T_1 : \exists H \text{ s.t. } H \text{ is a hyperedge in } \mathcal{H}',\ H \in \lambda'(q) \text{ and } V \in H\}$ coincides with $\mathcal{P}_1$, and we are done. So suppose that V occurs in the hyperedge G. But then, by our construction, $\mathcal{P}' = \mathcal{Q}_i(QD_1) \cup \mathcal{P}_1$ holds, where both of the sets $\mathcal{Q}_i(QD_1)$ and $\mathcal{P}_1$ are connected subtrees of T_1 containing the root r_1 of T_1. Thus, $\mathcal{P}'$ is also connected.

Case 2: Suppose that $\ell = i$. It suffices to show the following fact: *Let q be an arbitrary Q-node that covers V and let $q_0, q_1, \ldots, q_m$ for some $m \geq 0$ denote the path in QD' from the root $r_1 = q_0$ of T_1 to the Q-node $q = q_m$. Then every Q-node q_α along this path covers V.*

Of course, the root $r_1 = q_0$ covers V, since $\lambda'(r_1)$ contains the hyperedge G with t-label j, which (by the $\eta_{i,j}$ operation) is adjacent to the H-vertex V in $\mathcal{I}'$. For the other Q-nodes q_α with $\alpha > 0$, we know that they can no longer contain the H-vertex V itself in their Q-label $\lambda'(q_\alpha)$. Hence, in particular, $\lambda'(q_m)$ contains some hyperedge H s.t. the H-vertex V occurs in H. Similarly to Case 2 in the proof of condition 2 above, we have to distinguish two subcases depending on whether the t-label of H equals j or not.

(i) If H has the t-label j, then we know by condition 2 that every Q-node q_α on the path from q_m to the root r_1 of QD' contains some hyperedge H' with t-label j in its Q-label $\lambda'(q_\alpha)$. Moreover (by the $\eta_{i,j}$ operation), the hyperedge H' in $\mathcal{H}'$ contains all H-vertices with t-label i. Hence, in this case, every such Q-node q_α in QD' indeed covers V.

(ii) On the other hand, suppose that the t-label of H is different from j. Then no H-vertex is added to this hyperedge by the $\eta_{i,j}$ operation. Hence, V already

occurred in the corresponding hyperedge H^- of the hypergraph $\mathcal{H}_1$ and also the Q-label $\lambda_1(q_m)$ in QD_1 already contained the hyperedge H^-. Hence, we may apply condition 3 of the induction hypothesis; i.e., either V occurs in some hyperedge F^- in $\mathcal{H}_1$ s.t. $F^- \in \lambda_1(q_{m-1})$ or $\lambda_1(q_{m-1})$ contains some I-node N whose t-label is i. In the former case, $F \in \lambda'(q_{m-1})$ and, therefore, q also covers V in QD'. In the latter case, N is replaced by G in $\lambda'(q_{m-1})$ and, hence, again q covers V in QD'. By an easy induction argument, we can show that in fact every Q-node $q_{m-\beta}$ with $\beta \in \{1, \ldots, m-1\}$ covers V in QD'.

Renaming of labels. Let $t' = \rho_{i \to j}(s_1)$. Moreover, let $\mathcal{J}_1$ be the subgraph of $\mathcal{I}$ defined by s_1 and let $\mathcal{H}_1$ be the corresponding hypergraph. By the induction hypothesis, there exists a query decomposition QD_1 of $\mathcal{H}_1$ for which the above three conditions hold. We claim that then QD_1 is also the desired query decomposition QD' of $\mathcal{H}'$. Note that by the renaming of labels, no new vertices or edges are introduced in $\mathcal{I}'$. Hence, $\mathcal{H}'$ is identical to $\mathcal{H}_1$ and, therefore, QD_1 is clearly a query decomposition of $\mathcal{H}'$, i.e., condition 1 holds.

The sets of Q-nodes $\mathcal{Q}_\ell(QD')$ with $\ell \in \{1, \ldots, k\}$ are obtained from the sets $\mathcal{Q}_\ell(QD_1)$ in the following way:

$$\mathcal{Q}_\ell(QD') = \begin{cases} \emptyset & \text{if } \ell = i, \\ \mathcal{Q}_i(QD_1) \cup \mathcal{Q}_j(QD_1) & \text{if } \ell = j, \\ \mathcal{Q}_\ell(QD_1) & \text{otherwise.} \end{cases}$$

Hence, condition 2 also follows immediately from the induction hypothesis.

As for condition 3, let p be a Q-node in QD' and let q be the parent of p in QD'. Moreover, let V be an H-vertex with t-label ℓ s.t. p covers V. The only interesting case is that the t-label of V was changed from i to j by the $\rho_{i \to j}$ operation. But then, by the induction hypothesis, either q covers V in QD_1 or the Q-label $\lambda_1(q)$ contains some I-node N whose t-label (in s_1) is i. In the former case, q still covers V in QD' and in the latter case the Q-label $\lambda'(q)$ contains the I-node N, which now has the t-label j. Hence, in either case, condition 3 holds. □

The converse of Theorem 3.1 is clearly not true. This is due to the fact that the clique-width is a hereditary property with respect to (w.r.t.) induced subgraphs (and any subhypergraph $\mathcal{H}'$ of a hypergaph $\mathcal{H}$ indeed gives rise to an induced subgraph $\mathcal{I}(\mathcal{H}')$ of the incidence graph $\mathcal{I}(\mathcal{H})$), whereas α-acyclicity, query-width, and hypertree-width are not. In particular, we can take any hypergraph $\mathcal{H}'$ with high clique-width and possibly high hypertree-width and transform it into the following hypergraph $\mathcal{H}$: Let H be a new hyperedge that contains all vertices of $\mathcal{H}'$ and let $\mathcal{H}$ be the result of adding H to $\mathcal{H}'$. Then $\mathcal{H}$ is α-acyclic and, therefore, $qw(\mathcal{H}) = hw(\mathcal{H}) = 1$ holds. On the other hand, the incidence graph of $\mathcal{H}'$ is an induced subgraph of the incidence graph of $\mathcal{H}$. Hence, the clique-width of $\mathcal{H}$ is at least as high as in the case of $\mathcal{H}'$.

The following example will help to illustrate the construction in the proof of Theorem 3.1.

Example. Consider the conjunctive query $A(x_1, x_2, x_3) \wedge B(x_2, x_3, x_4)$. The corresponding hypergraph $\mathcal{H}$ has 4 vertices, $\{V_1, V_2, V_3, V_4\}$, and 2 hyperedges, $H_1 = \{V_1, V_2, V_3\}$ and $H_2 = \{V_2, V_3, V_4\}$. The incidence graph $\mathcal{I}$ of $\mathcal{H}$ and the tree representation of a k-expression t (with $k = 3$) generating $\mathcal{I}$ are displayed in Figure 3.1.

Now let us traverse the tree representation of t bottom-up and see what the various subexpressions of t and the corresponding query decompositions look like. Actually, we shall discuss only the subexpressions s_i along the left-most path of the tree representation of t in detail. The corresponding query decompositions QD_i are depicted in Figure 3.2.

a) incidence graph $\mathcal{I}$: b) k-expression t:

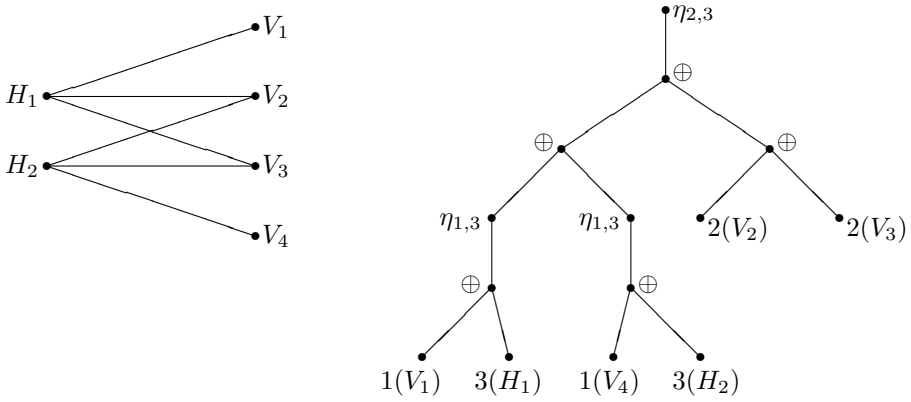


FIG. 3.1. *$\mathcal{I}$ and t.*

The query decomposition QD_1 corresponding to $s_1 = 1(V_1)$ consists of a single node labeled by V_1. Likewise, from $3(H_1)$ we get a query decomposition with a single node labeled by H_1. Hence, the labeling of the new root in the query decomposition QD_2 corresponding to $s_2 = 1(V_1) \oplus 3(H_1)$ contains both V_1 and H_1 from its child nodes, since they have different labels in s_2. In the query decomposition QD_3 obtained from $s_3 = \eta_{1,3}(s_2)$, we have to replace all occurrences of V_1 outside the root of QD_2 by the hyperedge H_1. Actually, in this case, this step was not really necessary. However, when we discuss the k-expression s_6 below, it will become clear why this replacement is, in general, required.

Now consider the query decomposition QD_4 corresponding to the k-expression $s_4 = s_3 \oplus \eta_{1,3}[1(V_4) \oplus 3(H_2)]$. The subtree in QD_4 corresponding to $\eta_{1,3}[1(V_4) \oplus 3(H_2)]$ is obtained analogously to the query decomposition QD_3 corresponding to s_3. Moreover, in the root of QD_4, we are only allowed to select one representative from the sets $\{V_1, V_4\}$ and $\{H_1, H_2\}$, respectively, since V_1 and V_4 (when considered as nodes in the graph generated by s_4), on the one hand, and H_1 and H_2, on the other hand, have the same label in s_4.

No new ideas are required for the construction of the query decomposition QD_5 corresponding to the k-expression $s_5 = s_4 \oplus (2(V_2) \oplus 2(V_3))$. Note that in the query decomposition QD_6 corresponding to $s_6 = t = \eta_{2,3}(s_5)$, it is indeed necessary to replace the occurrences of V_2 and V_3 by the hyperedge H_1. In particular, QD_5 is no longer a valid query decomposition after the $\eta_{2,3}$ operation has been applied to the incidence graph. This is due to the fact that after this operation, the hyperedge H_1 contains the vertex V_3 in the corresponding hypergraph. But then the connectedness condition would be violated by QD_5, since the root covers the vertex V_3 (since this vertex is now contained in H_1) and also the right-most leaf node of QD_5 covers the vertex V_3. However, in QD_5, the node lying in between them contains only V_2.

Recall from [29] that $qw(\mathcal{H}) \geq hw(\mathcal{H})$ holds for every hypergraph $\mathcal{H}$. Hence, by Theorem 3.1, we immediately get the following corollary.

COROLLARY 3.2 (hypertree-width is bounded by clique-width). *Let $\mathcal{H}$ be an arbitrary hypergraph with incidence graph $\mathcal{I}(\mathcal{H})$. Then $hw(\mathcal{H}) \leq cw(\mathcal{I}(\mathcal{H}))$ holds.*

Moreover, by the correspondence between queries and hypergraphs, we have the following.

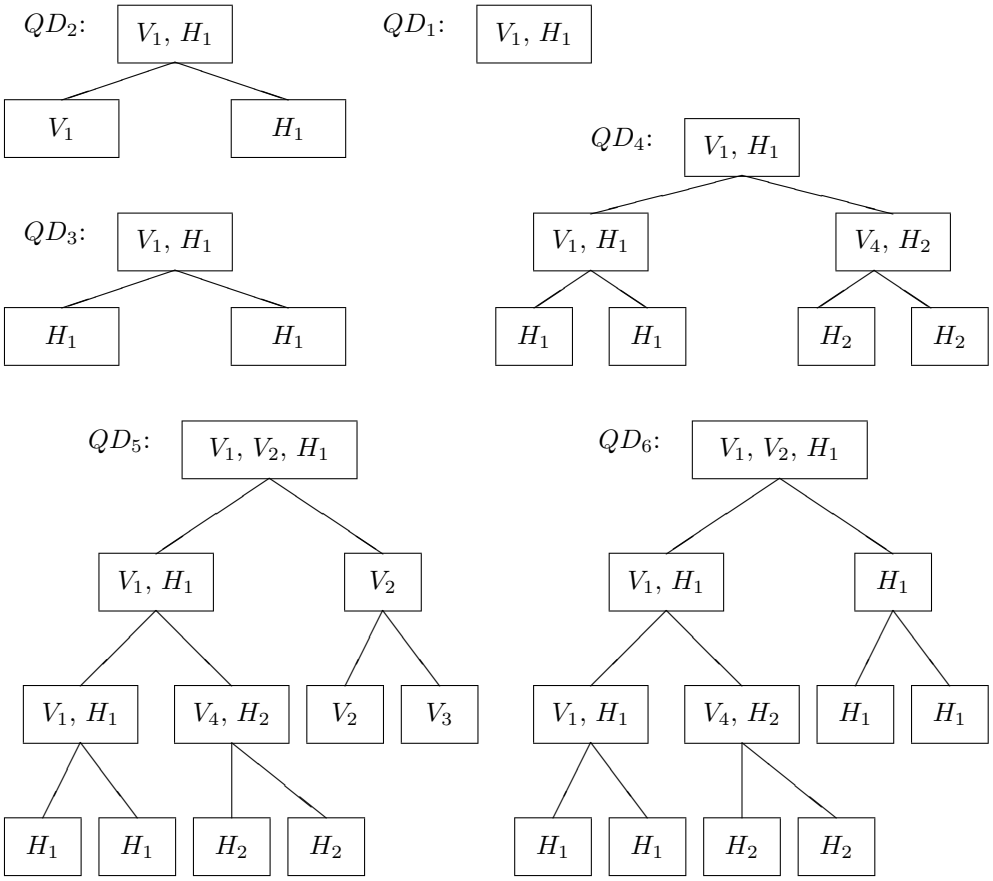


FIG. 3.2. *Query decompositions* $QD_1, \ldots, QD_6$.

COROLLARY 3.3 (width of a conjunctive query). *Let Q be a conjunctive query with incidence graph $\mathcal{I}(Q)$. Then $hw(Q) \leq qw(Q) \leq cw(\mathcal{I}(Q))$ holds.*

The above corollary has another interesting aspect: Apart from the special case of $k \leq 3$, it is not known whether graphs with clique-width $\leq k$ can be recognized in polynomial time for fixed k (cf. [14]). In contrast, conjunctive queries (or, equivalently, hypergraphs) with hypertree-width $\leq k$ actually can be recognized in polynomial time. In fact, this decision problem is highly parallelizable since it is even contained in the low complexity class LOGCFL (cf. [29]). In other words, apart from being the more general concept, bounded hypertree-width also has better properties as far as recognizing such conjunctive queries is concerned.

4. Restricting the form of the structures. In this section we consider monadic second-order queries over hypergraphs, where quantification is allowed over variables that stand for vertices or hyperedges. Moreover, there are two unary predicates P_V, P_H and a binary predicate *edg* with the following meaning: $P_V(x)$, $P_H(x)$ state that the argument x is a vertex or a hyperedge, respectively, of the hypergraph. By $edg(v, h)$ we can express that the vertex v is contained in the hyperedge h. Clearly,

these formulae correspond to MS_1 formulae that are evaluated over the incidence graphs (when considered as a labeled graph with two labels) of hypergraphs. Thus, the evaluation of such formulae is tractable if the incidence graphs under consideration are of bounded clique-width. From [16] we know that a class of labeled graphs with p labels (for fixed $p \geq 1$) is of bounded clique-width iff the same graphs without labels are of bounded clique-width. Hence, in what follows, we shall ignore the two different labels of the nodes of the incidence graph, since they have no effect on the tractability of the evaluation of MS_1 formulae.

It has already been mentioned that clique-width is a hereditary property, while α-acyclicity and hypertree-width are not. In case of restrictions on the queries, this does not matter. However, if we look for appropriate restrictions on the structures, then it can be easily verified that α-acyclicity or bounded hypertree-width will clearly not suffice to make the evaluation of any fixed MS_1 formula tractable. Instead, we shall consider β-acyclicity and β-hypertree-width here as well as γ-acyclicity, which is even slightly more restrictive than β-acyclicity. It will turn out that γ-acyclic hypergraphs have clique-width ≤ 3 and that the β-hypertree-width of any hypergraph is less than or equal to the clique-width (of the incidence graph). Hence, γ-acyclicity of a class of hypergraphs implies bounded clique-width (w.r.t. the incidence graph), which, in turn, implies bounded β-hypertree-width. In other words, γ-acyclicity and bounded β-hypertree-width can be considered as "lower and upper bounds," respectively, on the expressive power of the notion of bounded clique-width. However, we shall also show that bounded β-hypertree-width is not sufficient to ensure the tractability of the evaluation of an arbitrary but fixed MS_1 formula.

In the second part of this section, we shall have a brief look at the primal graph of hypergraphs.

4.1. Clique-width of the incidence graph. In [20], D'Atri and Moscarini provided an algorithm for recognizing γ-acyclic hypergraphs. (For details, see the original paper or [24].) In terms of the incidence graph of a hypergraph, we get an algorithm consisting of the following rules:

1. *Deletion of isolated nodes*: If a node N in $\mathcal{I}$ has no adjacent node, then N may be deleted.
2. *Deletion of "ear nodes"*: If a node N in $\mathcal{I}$ has exactly one adjacent node, then N may be deleted.
3. *Contraction of two-element modules*: If two nodes N and N' in $\mathcal{I}$ are adjacent to exactly the same nodes in $\mathcal{V} - \{N, N'\}$, then one of them may be deleted.

Moreover, we assume that an edge is deleted from the incidence graph if one of its endpoints is deleted by one of these rules. Then a hypergraph $\mathcal{H}$ is γ-acyclic iff the exhaustive, nondeterministic application of the above rules transforms the incidence graph $\mathcal{I}(\mathcal{H})$ into the empty graph.

As far as the clique-width (of the incidence graphs) of γ-acyclic hypergraphs is concerned, several known results can be combined to get the following.

THEOREM 4.1 (γ-acyclicity implies bounded clique-width). *Every γ-acyclic hypergraph has clique-width ≤ 3 (w.r.t. the incidence graph).*

Proof. (See [8].) Let $\mathcal{H}$ be an arbitrary γ-acyclic hypergraph. We know from [2] that the incidence graph $\mathcal{I}(\mathcal{H})$ is "(6,2)-chordal." Building upon a characterization of "distance-hereditary" graphs in [4], it was shown in [21] that a graph is bipartite, (6,2)-chordal iff it is bipartite, distance-hereditary. Finally, in [27], it is shown that any distance-hereditary graph has clique-width ≤ 3. □

When we define the notion of generalized tree-width in section 5, we shall revisit the algorithm from [27], which constructs a 3-expression of any given distance-hereditary graph. In particular, this algorithm can be used to compute a 3-expression for the incidence graph of a given γ-acyclic hypergraph. The example in section 5 for a modified version of this algorithm will also help to illustrate the above theorem.

In the remainder of this section, we compare the clique-width (of the incidence graph) with β-acyclicity and β-hypertree-width.

THEOREM 4.2 (clique-width of the incidence graph versus β-acyclicity). *The class of β-acyclic hypergraphs is of unbounded clique-width (w.r.t. the incidence graph).*

Proof. Consider the sequence $(\mathcal{H}_n)_{n\geq 1}$ of hypergraphs, where $\mathcal{H}_n$ has the vertices $V = \{y_1, \ldots, y_n\} \cup \{x_{ij} : 1 \leq i < j \leq n\}$ and n hyperedges $H_1, \ldots, H_n$ with

$$H_l = \{y_l\} \cup \{x_{\alpha\beta} : \alpha < \beta \leq l\} \cup \{x_{l\gamma} : l < \gamma \leq n\} \text{ for } l \in \{1, \ldots, n\};$$

i.e., $H_1 = \{y_1, x_{12}, x_{13}, \ldots, x_{1n}\}$, $H_2 = \{y_2, x_{12}, x_{23}, \ldots, x_{2n}\}$, $H_3 = \{y_3, x_{12}, x_{13}, x_{23}, x_{34}, \ldots, x_{3n}\}$, $\ldots$, $H_n = \{y_n, x_{12}, \ldots, x_{1n}, x_{23}, \ldots, x_{2n}, \ldots, x_{(n-1)n}\}$. The β-acyclicity of $\mathcal{H}_n$ can be shown via the following observation: Let H_i, H_j, H_k be hyperedges of $\mathcal{H}_n$ with $i < j < k$. Then the relation $H_i \cap H_j \subseteq H_k$ holds. Now let $\mathcal{H}'$ be a subhypergraph of $\mathcal{H}_n$ with vertices $V' \subseteq V$ and m hyperedges $H'_{i_1}, \ldots, H'_{i_m}$ s.t. $H_{i_1}, \ldots, H_{i_m}$ are hyperedges in $\mathcal{H}_n$ and $H'_{i_j} = H_{i_j} \cap V'$ holds for every $j \in \{1, \ldots, m\}$. W.l.o.g., let $1 \leq i_1 < i_2 < \cdots < i_m \leq n$. Then $\mathcal{H}'$ is α-acyclic, since a join tree of $\mathcal{H}'$ can be obtained by labeling the root node with H'_{i_m} and attaching $m-1$ child nodes with the labels $H'_{i_1}, \ldots, H'_{i_{(m-1)}}$ to the root.

It remains to prove that the incidence graphs $(\mathcal{I}_n)_{n\geq 1}$ of $(\mathcal{H}_n)_{n\geq 1}$ are of unbounded clique-width. In fact, we show that $cw(\mathcal{I}_n) \geq n$ holds for every $n \geq 2$. Suppose on the contrary that $\mathcal{I}_n$ is defined by some k-expression t with $k \leq n-1$. Moreover, let t' be a subexpression in t s.t. for some $i \in \{1, \ldots, n\}$ the nodes H_i and y_i as well as at least $n-2$ nodes from the set $\{x_{1i}, \ldots, x_{(i-1)i}, x_{i(i+1)}, \ldots, x_{in}\}$ (consisting of $n-1$ nodes) have already been introduced in t', and let t' be minimal with this property; i.e., no proper subexpression t'' of t' has this property. By the minimality of t', this subexpression t' is of the form $t' = r \oplus s$ for appropriately chosen k-expressions r and s. Then we derive a contradiction in the following way.

Fact 1. Suppose that two vertices H_α and H_β with $\alpha \neq \beta$ have been introduced by the k-expression t'. Then H_α and H_β have different labels in the graph generated by t'. This can be seen as follows: Suppose on the contrary that H_α and H_β have the same label. W.l.o.g., let $\alpha < \beta$. Then the vertices in $V_\beta = \{y_\beta\} \cup \{x_{1\beta}, \ldots, x_{(\alpha-1)\beta}, x_{(\alpha+1)\beta}, \ldots, x_{(\beta-1)\beta}, x_{\beta(\beta+1)}, \ldots, x_{\beta n}\}$ distinguish the vertices H_α and H_β. Hence, all of the edges connecting H_β with the vertices in V_β must already exist in t'. Thus, all of the vertices in $\{H_\beta\} \cup V_\beta$ must have already been introduced in one of the subexpressions r or s of t', which contradicts the minimality of t'.

Fact 2. Suppose that the vertices H_α and $x_{\beta\gamma}$ with arbitrary α, β, and γ have been introduced by the k-expression t'. Note that H_α is adjacent to all of the vertices in $V_\alpha = \{y_\alpha\} \cup \{x_{1\alpha}, \ldots, x_{(\alpha-1)\alpha}, x_{\alpha(\alpha+1)}, \ldots, x_{\alpha n}\}$, while $x_{\beta\gamma}$ is not. Hence, one can show analogously to Fact 1 above that H_α and $x_{\beta\gamma}$ have different labels.

Fact 3. If the vertices H_α and y_β with arbitrary α and β have been introduced by the k-expression t', then H_α and y_β have different labels, since H_α and y_β are also distinguished by the vertices in the above set V_α.

In order to conclude the proof, we define a set $\mathcal{S}$ of n vertices of t' s.t. these vertices have pairwise distinct labels in t'.

(i) H_i is in $\mathcal{S}$.

(ii) Let X denote those vertices in $\{x_{1i}, \ldots, x_{(i-1)i}, x_{i(i+1)}, \ldots, x_{in}\}$, which already exist in the k-expression t'. By assumption, X has at least $n-2$ elements. Now we traverse X from right to left. If the label of some $x_{\alpha\beta} \in X$ does not occur in X to the right of $x_{\alpha\beta}$, then we add $x_{\alpha\beta}$ to $\mathcal{S}$. Otherwise, suppose that $x_{\gamma\delta}$ is a vertex to the right of $x_{\alpha\beta}$ s.t. $x_{\alpha\beta}$ and $x_{\gamma\delta}$ have the same label. Then we distinguish two cases: If $\alpha < i$ and $\beta = i$ hold, then H_α distinguishes the nodes $x_{\alpha i}$ and $x_{\gamma\delta}$. Hence, H_α already exists in t' and we may add H_α to $\mathcal{S}$. Otherwise (i.e., $\alpha = i$ and $\beta > i$ hold), H_β distinguishes the nodes $x_{i\beta}$ and $x_{\gamma\delta}$. Hence, H_β already exists in t' and we add H_β to $\mathcal{S}$.

(iii) Finally we consider y_i. Let $X \subseteq \{x_{1i}, \ldots, x_{(i-1)i}, x_{i(i+1)}, \ldots, x_{in}\}$ be defined as above. If the label of y_i does not occur in X, then we add y_i to $\mathcal{S}$. Otherwise, let $x_{\alpha\beta}$ denote the right-most vertex in X that has the same label as y_i. Then we distinguish the same cases as above: If $\alpha < i$ and $\beta = i$ hold, then H_α distinguishes the nodes $x_{\alpha i}$ and y_i. Hence, we may add H_α to $\mathcal{S}$. Otherwise (i.e., $\alpha = i$ and $\beta > i$ hold), we add H_β to $\mathcal{S}$.

Then $\mathcal{S}$ contains n vertices. In particular, all of the vertices H_α and H_β that are added to $\mathcal{S}$ by the above construction are pairwise distinct. Moreover, by the construction of $\mathcal{S}$ and by Facts 1 through 3 above, all of the vertices in $\mathcal{S}$ have pairwise distinct labels. □

Actually, the above lower bound on the clique-width of the incidence graphs $(\mathcal{I}_n)_{n\geq 1}$ is quite tight. This follows from the fact that the incidence graphs $(\mathcal{I}_n)_{n\geq 1}$ are bipartite graphs, where one part of the partition has n nodes (namely, $\{H_1, \ldots, H_n\}$). Hence, as we have seen in the example in section 2.1, $\mathcal{I}_n$ has clique-width $\leq n + 2$.

Remark. Theorem 4.2 was shown independently in [8] as follows: It was shown in [2] that β-acyclic hypergraphs have bipartite, "chordal" incidence graphs. Moreover, we know from [9] that bipartite permutation graphs are a subclass of bipartite, chordal graphs. Finally, in [10] it was shown that the clique-width of bipartite permutation graphs is unbounded.

Now recall from Corollary 3.2 that the hypertree-width of any hypergraph $\mathcal{H}$ is less than or equal to the clique-width of the incidence graph of a hypergraph. Moreover, as has already been mentioned, the incidence graph $\mathcal{I}(\mathcal{H}')$ of any subhypergraph $\mathcal{H}'$ of $\mathcal{H}$ is an induced subgraph of $\mathcal{I}(\mathcal{H})$. Thus, $cw(\mathcal{I}(\mathcal{H}')) \leq cw(\mathcal{I}(\mathcal{H}))$ holds. We therefore immediately get the following corollary.

COROLLARY 4.3 (β-hypertree-width is bounded by clique-width). *Let $\mathcal{H}$ be a hypergraph with clique-width $= k$ (w.r.t. the incidence graph). Then $\mathcal{H}$ has β-hypertree-width $\leq k$.*

It has already been recalled from [29] that α-acyclic hypergraphs have hypertree-width $= 1$. Consequently, the β-acyclic hypergraphs have β-hypertree-width $= 1$. Thus, by Theorem 4.2 and Corollary 4.3, we know that bounded clique-width implies bounded β-hypertree-width, while the converse is not true. Now the question naturally arises as to whether bounded β-hypertree-width of the structures under consideration suffices to guarantee the tractability of the evaluation of any MS_1 formula. Unfortunately, the answer given in Theorem 4.5 below is negative. Thus, bounded clique-width remains the concept with the highest expressive power known so far s.t. MS_1 queries are still tractable (cf. Figure 1.1). The proof of this result will be based on the following lemma.

LEMMA 4.4 (β-hypertree-width of hypergraphs with big hyperedges). *Let $\mathcal{H} = \langle \mathcal{V}, \mathcal{E} \rangle$ be a hypergraph, where every hyperedge $H \in \mathcal{E}$ has at least $|\mathcal{V}| - 2$ vertices.*

Then the β-hypertree-width of $\mathcal{H}$ is ≤ 3.

Proof. Let $\mathcal{H}' = \langle \mathcal{V}', \mathcal{E}' \rangle$ be an arbitrary subhypergraph of $\mathcal{H}$, i.e., $\mathcal{H}'$ is obtained from $\mathcal{H}$ by deleting some hyperedges and/or vertices. Note that then $\mathcal{H}'$ also contains only "big" hyperedges; i.e., every hyperedge $H' \in \mathcal{E}'$ has at least $|\mathcal{V}'| - 2$ vertices.

We have to show that $\mathcal{H}'$ has a *hypertree decomposition* of width ≤ 3. In fact, we show that $\mathcal{H}'$ even has a *query decomposition* of width ≤ 3. Let $H \in \mathcal{E}'$ be an arbitrary hyperedge. Then $|\mathcal{V}' - H| \leq 2$ holds. Let $V_1, V_2 \in \mathcal{V}'$ s.t. $H \cup \{V_1, V_2\} = \mathcal{V}'$. Now we can construct a query decomposition QD of width ≤ 3 for $\mathcal{H}'$ as follows: Let the root r of QD be labeled with $\lambda(r) = \{H, V_1, V_2\}$. Moreover, for every $H' \in \mathcal{H}' - \{H\}$, we attach one child node p to r with label $\lambda(p) = \{H'\}$. □

THEOREM 4.5 (MS_1 queries and bounded β-hypertree-width). *The evaluation of an arbitrary fixed MS_1 query over a class $\mathcal{C}$ of hypergraphs is, in general, not tractable, even if $\mathcal{C}$ is of bounded β-hypertree-width.*

Proof. Let $\mathcal{G} = \langle V, E \rangle$ be an arbitrary graph and let $\mathcal{H} = \langle V, H \rangle$ be a hypergraph, where the set H of hyperedges is defined as follows: $H = \{V - \{x, y\} : \{x, y\}$ is an edge in $E\}$; i.e., every edge e of $\mathcal{G}$ is encoded by a hyperedge which contains all vertices of V except for the endpoints of e. Then every hyperedge of $\mathcal{H}$ has $|V| - 2$ vertices. Thus, by Lemma 4.4 above, this hypergraph $\mathcal{H}$ has β-hypertree-width at most 3. Moreover, the well-known NP-complete problem of graph-3-colorability can be expressed as an MS_1 query on the incidence graph of $\mathcal{H}$ (when considered as a labeled graph with two distinct labels for the nodes corresponding to vertices and hyperedges in $\mathcal{H}$, respectively; the unary predicates P_V and P_H refer to these labels) in the following way:

$$(\exists C_1)(\exists C_2)(\exists C_3)\ \text{“}C_1, C_2, \text{ and } C_3 \text{ provide a partition of } V\text{”}$$
$$\wedge\ (\forall x)(\forall y)[\,(P_V(x) \wedge P_V(y) \wedge (\exists h)(P_H(h) \wedge \neg edg(x,h) \wedge \neg edg(y,h)))$$
$$\rightarrow \text{“}x \text{ and } y \text{ have different colors”}].$$

Of course, the sentences "C_1, C_2, and C_3 provide a partition of V" and "x and y have different colors" can be easily expressed as MS_1 formulae, namely,

$(\forall x)[x \in C_1 \wedge x \notin C_2 \wedge x \notin C_3] \vee [x \in C_2 \wedge x \notin C_1 \wedge x \notin C_3] \vee [x \in C_3 \wedge x \notin C_1 \wedge x \notin C_2]$ and $[x \in C_1 \wedge y \notin C_1] \vee [x \in C_2 \wedge y \notin C_2] \vee [x \in C_3 \wedge y \notin C_3]$, respectively. □

4.2. Clique-width of the primal graph. In this section we compare the β-acyclicity and β-hypertree-width with clique-width of the primal graph. By Corollary 4.3, we know that bounded clique-width of the *incidence graph* implies bounded hypertree-width. It can be easily checked that this implication is no longer true if we consider the clique-width of the *primal graph* instead. This can be seen by inspecting the class of cliques, whose primal graphs (which coincide with the cliques themselves) have clique-width 2. On the other hand, the class of cliques is of unbounded tree-width and, therefore, also of unbounded β-hypertree-width, since—in contrast to *hypergraphs*—bounded tree-width and bounded hypertree-width coincide on *graphs*. This is due to the fact that, for every graph, a hypertree decomposition $\langle T, \chi, \lambda \rangle$ of width k corresponds to a tree decomposition $\langle T, \chi \rangle$ of width $\leq 2k$.

Now the question naturally arises as to whether bounded clique-width of the primal graphs allows for a strictly larger class of hypergraphs than bounded hypertree-width or at least than bounded β-hypertree-width. Below, we give a negative answer.

THEOREM 4.6 (clique-width of the primal graph versus β-acyclicity). *The class of β-acyclic hypergraphs is of unbounded clique-width w.r.t. the primal graphs.*

Proof. We consider again the class $(\mathcal{H}_n)_{n \geq 1}$ of β-acyclic hypergraphs of Theorem 4.2, where $\mathcal{H}_n$ has the vertices $V = \{y_1, \ldots, y_n\} \cup \{x_{ij} : 1 \leq i < j \leq n\}$ and the

hyperedges $H_1, \ldots, H_n$ with $H_l = \{y_l\} \cup \{x_{\alpha\beta} : \alpha < \beta \leq l\} \cup \{x_{l\gamma} : l < \gamma \leq n\}$ for $l \in \{1, \ldots, n\}$.

Similarly to Theorem 4.2, we show that the clique-width of the primal graph $\mathcal{P}_n$ of $\mathcal{H}_n$ increases with n. Note, however, that the situation here is a bit different from Theorem 4.2. In particular, $\mathcal{P}_n$ contains only nodes y_i and x_{ij}, but no H_i's. Moreover, the x_{ij}'s form a big clique in $\mathcal{P}_n$, since all of these vertices occur in the hyperedge H_n. Nevertheless, we can show that $cw(\mathcal{P}_n) \geq \frac{n-1}{2}$ holds for every $n \geq 2$.

Suppose on the contrary that $\mathcal{P}_n$ is defined by some k-expression t with $k < \frac{n-1}{2}$. Moreover, let t' be a subexpression in t s.t. for some $i \in \{1, \ldots, n\}$ the node y_i and at least $n-2$ nodes from the set $\{x_{1i}, \ldots, x_{(i-1)i}, x_{i(i+1)}, \ldots, x_{in}\}$ have already been introduced in t'. Moreover, let t' be minimal with this property. Then t' is again of the form $t' = r \oplus s$. We derive a contradiction in the following way.

Fact 1. Suppose that two vertices y_α and y_β with $\alpha \neq \beta$ have been introduced by the subexpression t' of t. Then y_α and y_β have different labels in the graph generated by t'. This is due to the fact that, for $\alpha < \beta$, the vertices in $X_\beta = \{x_{1\beta}, \ldots, x_{(\alpha-1)\beta}, x_{(\alpha+1)\beta}, \ldots, x_{(\beta-1)\beta}, x_{\beta(\beta+1)}, \ldots, x_{\beta n}\}$ distinguish the vertices y_α and y_β.

Fact 2. Let X_i be defined as $X_i = \{x_{1i}, \ldots, x_{(i-1)i}, x_{i(i+1)}, \ldots, x_{in}\}$. Moreover, suppose that $x_{\alpha i}$ and $x_{\gamma\delta}$ are nodes in X_i s.t. $\alpha < \gamma$ (i.e., $x_{\alpha i}$ occurs in X_i "to the left" of $x_{\gamma\delta}$) and both nodes already exist in t'. Then either $x_{\alpha i}$ and $x_{\gamma\delta}$ have distinct labels in t' or y_α already exists in t'. This follows immediately from the fact that $x_{\alpha i}$ is adjacent to y_α in $\mathcal{P}_n$, whereas $x_{\gamma\delta}$ is not.

Fact 3. Let X_i be defined as above and suppose that $x_{i\beta}$ and $x_{\gamma\delta}$ are vertices in X_i s.t. both vertices already exist in t' and $\beta < \delta$ holds (i.e., again $x_{i\beta}$ occurs in X_i "to the left" of $x_{\gamma\delta}$). Then either $x_{i\beta}$ and $x_{\gamma\delta}$ have distinct labels in t' or y_β already exists in t', since y_β is adjacent to $x_{i\beta}$ in $\mathcal{P}_n$ but not to $x_{\gamma\delta}$.

Now we define two sets $\mathcal{X}$ and $\mathcal{Y}$ of vertices which have already been introduced by t'. These two sets together will contain $n-1$ vertices in total. Moreover, we can show that the vertices contained in each of these sets have pairwise distinct labels.

(i) Initially, we set $\mathcal{X} := \emptyset$ and $\mathcal{Y} := \{y_i\}$.

(ii) Let X denote the set of those vertices in $\{x_{1i}, \ldots, x_{(i-1)i}, x_{i(i+1)}, \ldots, x_{in}\}$ that already exist in the k-expression t'. By assumption, there are $n-2$ such vertices. Now we traverse the elements in X from right to left. If the label of some $x_{\alpha\beta} \in X$ does not occur in X to the right of $x_{\alpha\beta}$, then we add $x_{\alpha\beta}$ to $\mathcal{X}$. Otherwise, suppose that $x_{\gamma\delta}$ is a vertex to the right of $x_{\alpha\beta}$ s.t. $x_{\alpha\beta}$ and $x_{\gamma\delta}$ have the same label. Then we distinguish two cases: If $\alpha < i$ and $\beta = i$ hold, then y_α already exists in the k-expression t' by Fact 2. Hence, we may add y_α to $\mathcal{Y}$. Otherwise (i.e., $\alpha = i$ and $\beta > i$ hold), y_β must already exist in t' by Fact 3 and we add y_β to $\mathcal{Y}$.

We have $|\mathcal{X}| + |\mathcal{Y}| = n - 1$. Thus, one of the sets $\mathcal{X}$ or $\mathcal{Y}$ must have at least $\frac{n-1}{2}$ vertices. Moreover, these vertices have pairwise distinct labels in t' by the construction (in the case of $\mathcal{X}$) and by Fact 1 above (in the case of $\mathcal{Y}$). □

By Theorem 4.6 above and the fact that β-acyclic hypergraphs have β-hypertree-width 1, we know that bounded β-hypertree-width does not necessarily imply bounded clique-width of the primal graph. Moreover, it has already been explained above that bounded clique-width of the primal graph does not necessarily imply bounded β-hypertree-width. We thus get the following result.

COROLLARY 4.7 (uncomparability). *The concepts of bounded β-hypertree-width of hypergraphs and bounded clique-width of the corresponding primal graphs are uncomparable; i.e., on the one hand, there exists a class $\mathcal{C}_1$ of hypergraphs s.t. the β-*

hypertree-width of the hypergraphs in $\mathcal{C}_1$ is bounded by some fixed constant k, while the corresponding class of primal graphs is of unbounded clique-width. On the other hand, there exists a class $\mathcal{C}_2$ of hypergraphs s.t. the β-hypertree-width of the hypergraphs in $\mathcal{C}_2$ is unbounded while the clique-width of the corresponding primal graphs is bounded by some fixed constant k.

5. Generalized tree-width. As was already mentioned in section 2, clique-width is much more powerful than tree-width. On the other hand, the lack of an efficient procedure for recognizing graphs with clique-width $\leq k$ for some arbitrary but fixed k is a major drawback of clique-width. Hence, it is worth trying to extend the notion of tree-width to some kind of "generalized tree-width," which is more powerful than tree-width and which is still efficiently recognizable. One such generalization is proposed below.

Recall from [23] that the existence of a big complete bipartite graph as a subgraph of a graph $\mathcal{G}$ has a very bad effect on the tree-width of $\mathcal{G}$, e.g., consider the sequence $(\mathcal{H}_n)_{n\geq 1}$ of hypergraphs, where $\mathcal{H}_n$ has vertices $V = \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\}$ and n hyperedges $H_1, \ldots, H_n$ with $H_i = \{y_i, x_1, \ldots, x_n\}$. Then, for every n, the (incidence graph $\mathcal{I}_n$ of the) hypergraph $\mathcal{H}_n$ has tree-width n, since it contains the complete bipartite graph with nodes $\{x_1, \ldots, x_n\}$ and $\{H_1, \ldots, H_n\}$, respectively. On the other hand, for every n, $\mathcal{H}_n$ is γ-acyclic; i.e., the simple transformations of the γ-acyclicity algorithm in [20] (recalled in section 4.1) suffice to reduce the incidence graph of $\mathcal{H}_n$ to the empty graph. In particular, the complete bipartite graph contained in the incidence graph of any such hypergraph can be eliminated by these simple transformations. It therefore makes sense to consider the following generalization of the tree-width.

DEFINITION 5.1 (generalized tree-width). *Let $\mathcal{G}$ be an arbitrary graph and let $\mathcal{G}'$ be the graph that results from exhaustive application of the following rules: deletion of isolated nodes, deletion of ear nodes, and contraction of two-element modules. Then we define the* generalized tree-width *of $\mathcal{G}$ as $gtw(\mathcal{G}) = tw(\mathcal{G}')$.*

In order to make sure that $gtw(\mathcal{G})$ is well defined, we need the following property of the above transformation rules.

PROPOSITION 5.2. *Let $\mathcal{G}$ be an arbitrary graph and let $\mathcal{G}'$ and $\mathcal{G}''$ be graphs that result from exhaustive application of the following rules: deletion of isolated nodes, deletion of ear nodes, and contraction of two-element modules. Then $\mathcal{G}'$ and $\mathcal{G}''$ are isomorphic.*

Proof. The number of possible applications of the above transformation rules is clearly finite. Hence, by general considerations on rewrite systems (cf. [3, 22]), it suffices to prove the following "local confluence" property: Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be graphs that can be obtained from some graph $\mathcal{G}$ via a single rule application. Then there exist graphs $\mathcal{G}_1'$ and $\mathcal{G}_2'$ s.t. $\mathcal{G}_1'$ and $\mathcal{G}_2'$ are isomorphic and $\mathcal{G}_i$ can be transformed into $\mathcal{G}_i'$ (for $i \in \{1, 2\}$) via finitely many applications of the above transformation rules. This can be easily shown by a case distinction over all 3×3 possibilities of rules that lead to $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively; e.g., suppose that $\mathcal{G}_1$ is obtained from $\mathcal{G}$ via the first rule (i.e., deletion of isolated nodes) and that $\mathcal{G}_2$ is obtained from $\mathcal{G}$ via the third rule (i.e., contraction of two-element modules). Moreover, let N_i with $i \in \{1, 2\}$ denote the node that is deleted from $\mathcal{G}$ in order to arrive at $\mathcal{G}_i$. Then $\mathcal{G}_1' = \mathcal{G}_2'$ is simply obtained from $\mathcal{G}_1$ by deleting also N_2 (via the third rule) and by deleting N_1 from $\mathcal{G}_2$ (via the first rule), respectively. The remaining cases are handled similarly. □

A polynomial time algorithm for recognizing the graphs with $gtw \leq k$ for some fixed k can be constructed in the obvious way, namely: First, an input graph $\mathcal{G}$ is

transformed into $\mathcal{G}'$ via the transformation given in Definition 5.1 above. Then we can apply the algorithm of [6], which decides in linear time whether $tw(\mathcal{G}') \leq k$ holds.

For our considerations on the tractability of evaluating MS_1 formulae, we are ultimately interested in the relationship between generalized tree-width and clique-width. In Theorem 5.7 we shall show that bounded generalized tree-width implies bounded clique-width. For the proof of this result, we have to revisit and appropriately modify the algorithm in [27] for constructing a 3-expression of an arbitrary distance-hereditary graph.

The notions of "pruning sequence" and "pruning tree" are central to the algorithm in [27]. By slightly modifying these notions for our purposes here, we get the following definitions.

DEFINITION 5.3 (pruning sequence). *Let $\mathcal{G}_0 = \mathcal{G}$, $\mathcal{G}_1, \ldots, \mathcal{G}_n = \mathcal{G}'$ be a sequence of graphs s.t., for every $i \geq 1$, $\mathcal{G}_i$ is obtained from $\mathcal{G}_{i-1}$ by deleting some node V_{α_i} via one of the following rules: deletion of isolated nodes, deletion of ear nodes, and contraction of two-element modules. Then $S = s_1, \ldots, s_n$ is called a* pruning sequence *from $\mathcal{G}$ to $\mathcal{G}'$, where s_i with $1 \leq i \leq n$ is defined as follows:*[3]

(i) *If V_{α_i} is an isolated node in $\mathcal{G}_{i-1}$, then $s_i = \langle (V_{\alpha_i}, *), isolated \rangle$.*

(ii) *If V_{α_i} is an ear node in $\mathcal{G}_{i-1}$ s.t. its only neighbor is V_{β_i}, then $s_i = \langle (V_{\alpha_i}, V_{\beta_i}), ear \rangle$.*

(iii) *If V_{α_i} and V_{β_i} form a two-element module in $\mathcal{G}_{i-1}$ s.t. V_{α_i} and V_{β_i} are adjacent, then $s_i = \langle (V_{\alpha_i}, V_{\beta_i}), true \rangle$. In this case, V_{α_i} is called a* true twin *of V_{β_i} in $\mathcal{G}_{i-1}$.*

(iv) *If V_{α_i} and V_{β_i} form a two-element module in $\mathcal{G}_{i-1}$ s.t. V_{α_i} and V_{β_i} are not adjacent, then $s_i = \langle (V_{\alpha_i}, V_{\beta_i}), false \rangle$. In this case, V_{α_i} is called a* false twin *of V_{β_i} in $\mathcal{G}_{i-1}$.*

DEFINITION 5.4 (pruning trees). *Let $\mathcal{G}_0 = \mathcal{G}, \mathcal{G}_1, \ldots, \mathcal{G}_n = \mathcal{G}'$ be a sequence of graphs with $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$. Moreover, let $S = s_1, \ldots, s_n$ be a pruning sequence from $\mathcal{G}$ to $\mathcal{G}'$ and let $\mathcal{V}'' \subseteq \mathcal{V}$ be defined as*

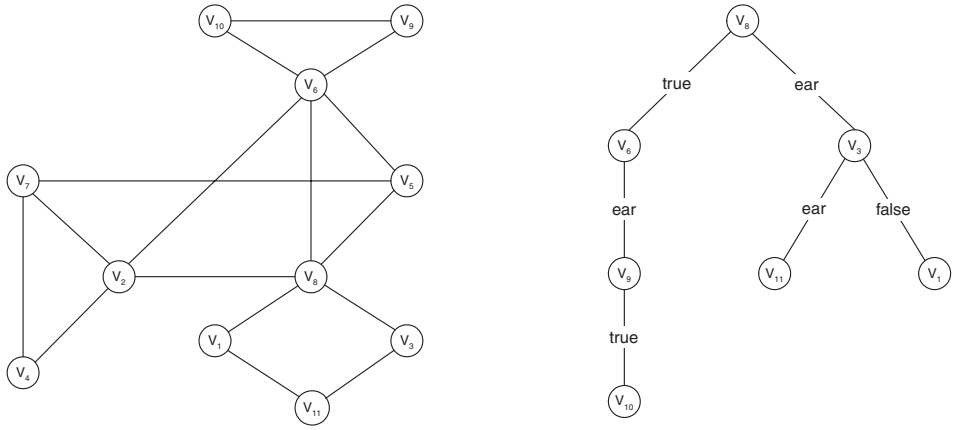
$$\begin{aligned} \mathcal{V}'' = &\{V_{\alpha_i} \mid s_i = \langle (V_{\alpha_i}, *), isolated \rangle\} \\ &\cup \{V_{\beta_i} \mid s_i = \langle (V_{\alpha_i}, V_{\beta_i}), x \rangle \text{ with } x \in \{ear, true, false\} \text{ and } V_{\beta_i} \in \mathcal{V}'\}; \end{aligned}$$

i.e., $\mathcal{V}''$ contains those nodes from $\mathcal{V}$ that were eventually deleted as isolated nodes plus those nodes which were "responsible" for the deletion of other nodes (i.e., of ear nodes or true/false twins) without ever being deleted themselves.

Now let $m = |\mathcal{V}''|$. Moreover, w.l.o.g., we assume that $\mathcal{V}'' = \{V_1, \ldots, V_m\}$. Then there exist m pruning trees *$\mathcal{T}_1, \ldots, \mathcal{T}_m$ corresponding to the pruning sequence S. These pruning trees are directed, edge-labeled trees. They are obtained by the algorithm* CONSTRUCT-PRUNING-TREES *given below.*

ALGORITHM CONSTRUCT-PRUNING-TREES.
begin
/* initialization of the pruning trees */
  for $i := 1$ **to** m **do**
    $\mathcal{T}_i :=$ the tree consisting of the root node V_i only;
  od;
/* adding the nodes deleted from $\mathcal{G}$ to the pruning trees */

[3] In [27] and [32], emphasis is put on the introduction of nodes rather than on their deletion. Hence, in the former papers, the order of the elements in a pruning sequence is reversed w.r.t. the definition here. However, the pruning trees defined next have essentially the same form as in [27].



(a) graph $\mathcal{G}$ (b) pruning tree $\mathcal{T}$

FIG. 5.1. *$\mathcal{G}$ and $\mathcal{T}$.*

for $i := n$ **downto** 1 **do**
if $s_i = \langle (V_{\alpha_i}, V_{\beta_i}), ear \rangle$ **then** append V_{α_i} as right-most child to V_{β_i} and label the edge from V_{β_i} to V_{α_i} as "ear";
elsif $s_i = \langle (V_{\alpha_i}, V_{\beta_i}), true \rangle$ **then** append V_{α_i} as right-most child to V_{β_i} and label the edge from V_{β_i} to V_{α_i} as "true";
elsif $s_i = \langle (V_{\alpha_i}, V_{\beta_i}), false \rangle$ **then** append V_{α_i} as right-most child to V_{β_i} and label the edge from V_{β_i} to V_{α_i} as "false";
fi;
od;
end.

This algorithm is illustrated by the following example.

Example. Let the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be defined as $\mathcal{V} = \{V_1, \ldots, V_{11}\}$ and $\mathcal{E} = \{\{V_1, V_8\}, \{V_1, V_{11}\}, \{V_2, V_4\}, \{V_2, V_7\}, \{V_2, V_8\}, \{V_3, V_8\}, \{V_3, V_{11}\}, \{V_4, V_7\}, \{V_5, V_6\}, \{V_5, V_7\}, \{V_5, V_8\}, \{V_6, V_8\}, \{V_6, V_9\}, \{V_6, V_{10}\}, \{V_9, V_{10}\}\}$. The graph $\mathcal{G}$ is displayed in Figure 5.1(a).

The nodes $\{V_1, V_3, V_6, V_9, V_{10}, V_{11}\}$ can be deleted via the following pruning sequence: $\langle (V_1, V_3), false \rangle$, $\langle (V_{11}, V_3), ear \rangle$, $\langle (V_3, V_8), ear \rangle$, $\langle (V_{10}, V_9), true \rangle$, $\langle (V_9, V_6), ear \rangle$, $\langle (V_6, V_8), true \rangle$. We thus get the graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ with $\mathcal{V}' = \{V_2, V_4, V_5, V_7, V_8\}$ and $\mathcal{E}' = \{\{V_2, V_4\}, \{V_2, V_7\}, \{V_2, V_8\}, \{V_4, V_7\}, \{V_5, V_7\}, \{V_5, V_8\}\}$. Note that no isolated node was thus deleted. Moreover, V_8 is the only node in $\mathcal{G}'$ that was responsible for the deletion of other nodes. Hence, by the algorithm CONSTRUCT-PRUNING-TREES, only one pruning tree $\mathcal{T}$ can be constructed, which is depicted in Figure 5.1(b).

In [27], several fundamental properties of pruning trees are proven. In particular, it is shown how a pruning tree $\mathcal{T}$ can be used to construct a 3-expression for the subgraph of $\mathcal{G}$ that is induced by the nodes of $\mathcal{T}$. Before we come to this algorithm, we recall the following terminology from [27]: For a node V in the pruning tree $\mathcal{T}$, we write $\mathcal{T}_V$ to denote the set of nodes of the subtree of $\mathcal{T}$ rooted at V. Moreover, we call a node U in $\mathcal{T}_V$ a *twin descendant* of V iff either $V = U$ or all the edges along the path from V to U are labeled with "true" or "false." Finally, by $\mathcal{G}[\mathcal{T}_V]$ we denote the subgraph of $\mathcal{G}$ that is induced by the nodes in $\mathcal{T}_V$.

LEMMA 5.5 (pruning tree and adjacency). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ be*

graphs and let S be a pruning sequence from $\mathcal{G}$ to $\mathcal{G}'$. Moreover, let $\mathcal{T}$ with root V be a pruning tree that is obtained from S via the algorithm CONSTRUCT-PRUNING-TREES. *Finally let U and U' be nodes in $\mathcal{V}$ s.t. U is in $\mathcal{T}$ and U' is outside $\mathcal{T}$. Then U and U' are adjacent in $\mathcal{G}$ iff V is adjacent to U' in $\mathcal{G}$ and U is a twin descendant of V.*

Proof. The proof is implicit in the proof of Lemma 3.7 in [27]. □

Example. Recall the graph $\mathcal{G}$ and the pruning tree $\mathcal{T}$ with root V_8 in Figure 5.1. The only twin descendants of V_8 are the nodes V_6 and V_8 itself. On the one hand, the nodes V_6 and V_8 are indeed adjacent to exactly the same nodes outside $\mathcal{T}$, namely, V_2 and V_5. On the other hand, the other nodes in $\mathcal{T}$ (namely, $V_1, V_3, V_9, V_{10}, V_{11}$) are not adjacent to any node outside $\mathcal{T}$.

LEMMA 5.6 (3-expressions). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ be graphs and let S be a pruning sequence from $\mathcal{G}$ to $\mathcal{G}'$. Moreover, let $\mathcal{T}$ denote a pruning tree that is obtained from S via the algorithm* CONSTRUCT-PRUNING-TREES. *Finally, let V be an arbitrary node in $\mathcal{T}$ with child nodes $U_1, \ldots, U_l$ from left to right. Then there exists a 3-expression t_V that defines the graph $\mathcal{G}[\mathcal{T}_V]$ s.t. all twin descendants of V are labeled with 2 in $\mathcal{G}[\mathcal{T}_V]$ and all other nodes in $\mathcal{T}_V$ are labeled with 1. Such a 3-expression can be computed inductively by the algorithm* CONSTRUCT-3-EXPRESSION *given below.*

Proof. See Claim 3.9 in [27]. □

ALGORITHM CONSTRUCT-3-EXPRESSION.

begin
  $t := 2(V)$;
  let $U_1, \ldots, U_l$ denote the child nodes of V from left to right;
  for $i := l$ **downto** 1 **do**
    if the edge from V to U_i is labeled with "ear" **then**
      $t := \rho_{3\to 1}(\eta_{2,3}(\rho_{2\to 3}(t_{U_i}) \oplus t))$;
    if the edge from V to U_i is labeled with "true" **then**
      $t := \rho_{3\to 2}(\eta_{2,3}(\rho_{2\to 3}(t_{U_i}) \oplus t))$;
    if the edge from V to U_i is labeled with "false" **then**
      $t := t_{U_i} \oplus t$;
    fi;
  od;
  $t_v := t$;
end.

Note that if the pruning sequence from $\mathcal{G}$ to $\mathcal{G}'$ contains the deletion of an isolated node V, then the subgraph $\mathcal{G}[\mathcal{T}_V]$ of $\mathcal{G}$ is a distance-hereditary connected component of $\mathcal{G}$. As has already been noted in section 4.1, the incidence graphs of γ-acyclic hypergraphs are precisely the distance-hereditary, bipartite graphs. Hence, the above algorithm can be used to compute the 3-expression of (every connected component of) the incidence graph of any γ-acyclic hypergraph. Moreover, it is now also clear why we never had to delete an isolated node in the example above, since the graph $\mathcal{G}$ of Figure 5.1(a) consists of a single connected component, and this connected component is not distance-hereditary.

Example. We put the algorithm CONSTRUCT-3-EXPRESSION to work by continuing the above example. The 3-expressions t_V for the nodes V in the pruning tree $\mathcal{T}$ of Figure 5.1(b) are obtained as follows. As a shorthand notation, we shall write t_i to denote the 3-expression t_{V_i} for any i.

$$t_1 = 2(V_1),$$

$$t_{11} = 2(V_{11}),$$

$$t_3 = \rho_{3\to 1}(\eta_{2,3}(\rho_{2\to 3}(t_{11}) \oplus (t_1 \oplus 2(V_3)))),$$

$$t_{10} = 2(V_{10}),$$

$$t_9 = \rho_{3\to 2}(\eta_{2,3}(\rho_{2\to 3}(t_{10}) \oplus 2(V_9))),$$

$$t_6 = \rho_{3\to 1}(\eta_{2,3}(\rho_{2\to 3}(t_9) \oplus 2(V_6))),$$

$$t_8 = \rho_{3\to 2}(\eta_{2,3}(\rho_{2\to 3}(t_6) \oplus \rho_{3\to 1}(\eta_{2,3}(\rho_{2\to 3}(t_3) \oplus 2(V_8))))).$$

We are now ready to prove the following desirable property of the generalized tree-width.

THEOREM 5.7 (bounded generalized tree-width implies bounded clique-width). *Let $\mathcal{C}$ be a class of graphs and let k be some fixed constant s.t. $gtw(\mathcal{G}) \leq k$ for all $\mathcal{G} \in \mathcal{C}$. Then there exists a constant k' s.t. $cw(\mathcal{G}) \leq k'$ for all $\mathcal{G} \in \mathcal{C}$.*

Proof. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an arbitrary graph and let $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ be the graph obtained from $\mathcal{G}$ by exhaustive application of the rules in Definition 5.1 (i.e., deletion of isolated nodes, deletion of ear nodes, and contraction of two-element modules). Moreover, let $gtw(\mathcal{G}) \leq k$; i.e., the condition $tw(\mathcal{G}') \leq k$ holds. Recall from [19] that then we have $cw(\mathcal{G}') \leq \tau$ with $\tau = 2^{k+1} + 1$. It suffices to show that then also $cw(\mathcal{G}) \leq k'$ with $k' = \tau + 1$ holds.

Let t' be a τ-expression that generates $\mathcal{G}'$. We assume that t' uses the τ labels $\{2, \ldots, k'\}$. Let S be a pruning sequence from $\mathcal{G}$ to $\mathcal{G}'$ and let $\mathcal{T}_1, \ldots, \mathcal{T}_m$ denote the corresponding pruning trees with root nodes $V_1, \ldots, V_m$. Moreover, for every $i \in \{1, \ldots, m\}$, let t_i denote the 3-expression obtained by the above algorithm CONSTRUCT-3-EXPRESSION for the subgraph $\mathcal{G}[\mathcal{T}_i]$ of $\mathcal{G}$. From these 3-expressions $t_1, \ldots, t_m$ together with t', we construct a k'-expression s for $\mathcal{G}$ as follows.

First we transform t' into t'' by the following replacement steps: For every V_i with $i \in \{1, \ldots, m\}$ s.t. V_i is still contained in $\mathcal{G}'$, we know that V_i is eventually introduced in t' by some subexpression of the form $\ell(V_i)$. By Lemma 5.6, V_i has the label 2 in t_i. Then we replace the subexpression $\ell(V_i)$ in t' either by t_i (if $\ell = 2$) or by $\rho_{2\to\ell}(t_i)$ (if $\ell \neq 2$).

Finally, all 3-expressions in $\{t_1, \ldots, t_m\}$, whose root node does not occur in $\mathcal{G}'$ anymore, are added to t'' via the $\oplus$-operator. We thus set

$$s := t'' \oplus \bigoplus_{V_i \notin \mathcal{V}'} t_i.$$

Obviously, t' is thus transformed into a k'-expression s using the labels $\{1, \ldots, k'\}$. It remains to show that s indeed generates the original graph $\mathcal{G}$.

As far as the nodes are concerned, every node in $\mathcal{V}$ either is introduced by some 3-expression t_i or is still contained in $\mathcal{V}'$. In the latter case, it is introduced by t'. Hence, by construction, s indeed introduces every node $V \in \mathcal{V}$. On the other hand, no node of $\mathcal{G}$ is introduced twice in s. This is due to the fact that any two distinct pruning trees $\mathcal{T}_i$ and $\mathcal{T}_j$ have no nodes from $\mathcal{V}$ in common, and for any pruning tree $\mathcal{T}_i$, only the root node of $\mathcal{T}_i$ can be contained in $\mathcal{V}'$.

As far as the edges of $\mathcal{G}$ are concerned, we know that each 3-expression t_i clearly introduces all edges in the subgraph $\mathcal{G}[\mathcal{T}_i]$ of $\mathcal{G}$. Likewise, t' introduces the edges in $\mathcal{E}'$. Hence, in order to show that s indeed introduces all edges $\{U, V\}$ it remains only to consider the following cases:

(i) Suppose that the nodes U and V occur in two distinct pruning trees $\mathcal{T}_i$ and $\mathcal{T}_j$, respectively. Moreover, let V_i and V_j denote the root nodes of these pruning trees. It follows from Lemma 5.5 that then V_i and V_j are also adjacent. Moreover, V_i and V_j must still be contained in $\mathcal{V}'$. Hence, this edge $\{V_i, V_j\}$ is eventually introduced by t' and thus by s. Moreover, by our construction of s and by Lemma 5.6, we know that U has the same label ℓ as V_i when the subexpression $\ell(V_i)$ in t' is replaced by t_i (if $\ell = 2$) or by $\rho_{2\to\ell}(t_i)$ (if $\ell \neq 2$). Likewise, U gets the same label as V_j. Hence, when eventually the edge between V_i and V_j is introduced in s, then the edge between U and V will be introduced as well.

(ii) Suppose that the node U occurs in some pruning tree $\mathcal{T}_i$ with root node V_i and that V is contained in $\mathcal{V}'$. By the same considerations as above, we know from Lemma 5.5 that then V_i and V are also adjacent. Moreover, U is introduced with the same label ℓ as V_i when the subexpression $\ell(V_i)$ in t' is replaced by t_i or by $\rho_{2\to\ell}(t_i)$, respectively. Hence, when eventually the edge between V_i and V is introduced in s, then the edge between U and V will be introduced as well.

Now consider also the opposite direction; i.e., we have to show that all edges $\{U, V\}$ introduced by s indeed exist in $\mathcal{G}$. Again, if both nodes U and V are in the same pruning tree $\mathcal{T}_i$ or if both nodes are left in $\mathcal{V}'$, then this is obviously the case. As above, it remains to consider the following two cases:

(i) Suppose that the nodes U and V occur in two distinct pruning trees $\mathcal{T}_i$ and $\mathcal{T}_j$, respectively. Moreover, let V_i and V_j denote the root nodes of these pruning trees. In order to introduce the edge $\{U, V\}$, the expression s and, therefore, t' actually introduces the edge $\{V_i, V_j\}$. Hence, V_i and V_j are indeed adjacent in $\mathcal{G}$. Moreover, U must have the same label as V_i in t_i. Likewise, V and V_j have the identical labels in t_j. Hence, by Lemma 5.6, U is a twin descendant of V_i in the pruning tree $\mathcal{T}_i$ and V is a twin descendant of V_j in $\mathcal{T}_j$. Thus, by Lemma 5.5, U and V are indeed adjacent in $\mathcal{G}$.

(ii) Suppose that the node U occurs in some pruning tree $\mathcal{T}_i$ with root node V_i and that V is contained in $\mathcal{V}'$. By the same considerations as above, we may conclude by Lemmas 5.5 and 5.6 that V_i and V are indeed adjacent, that V_i is a twin descendant of V_i in the pruning tree $\mathcal{T}_i$, and, finally, that U and V are indeed adjacent in $\mathcal{G}$.

In other words, the k'-expression s introduces exactly the nodes in $\mathcal{V}$ and exactly the edges in $\mathcal{E}$. Hence, s is indeed the desired k'-expression. □

Example. Let $\mathcal{G}$ and $\mathcal{G}'$ be the graphs from the example above. The graph $\mathcal{G}$ together with the pruning tree $\mathcal{T}$ are shown in Figure 5.1. Recall that $\mathcal{G}'$ has the form $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ with $\mathcal{V}' = \{V_2, V_4, V_5, V_7, V_8\}$ and $\mathcal{E}' = \{\{V_2, V_4\}, \{V_2, V_7\}, \{V_2, V_8\}, \{V_4, V_7\}, \{V_5, V_7\}, \{V_5, V_8\}\}$. A 4-expression t' of $\mathcal{G}'$ can be constructed as follows.

$$(\mathit{introduce}\ V_4)\ s_1 := 2(V_4),\ (\mathit{introduce}\ V_2)\ s_2 := 3(V_2) \oplus s_1,$$

$$(\mathit{connect}\ V_2, V_4)\ s_3 := \eta_{2,3}(s_2),\ (\mathit{introduce}\ V_7)\ s_4 := 4(V_7) \oplus s_3,$$

$$(\mathit{connect}\ V_4, V_7)\ s_5 := \eta_{2,4}(s_4),\ (\mathit{connect}\ V_2, V_7)\ s_6 := \eta_{3,4}(s_5),$$

$$(\mathit{introduce}\ V_8)\ s_7 := 5(V_8) \oplus s_6,\ (\mathit{connect}\ V_2, V_8)\ s_8 := \eta_{3,5}(s_7),$$

$$(\mathit{relabel}\ V_2)\ s_9 := \rho_{3\to 2}(s_8),\ (\mathit{introduce}\ V_5)\ s_{10} := 3(V_5) \oplus s_9,$$

$$(\mathit{connect}\ V_5, V_7)\ s_{11} := \eta_{3,4}(s_{10}),\ (\mathit{connect}\ V_5, V_8)\ s_{12} := \eta_{3,5}(s_{11}).$$

Then s_{12} is the desired 4-expression t' using the labels $\{2, \ldots, 5\}$.

Now recall that the 3-expression t of $\mathcal{G}[\mathcal{T}]$ has already been computed above. The 5-expression s that generates $\mathcal{G}$ can be constructed according to the proof of Theorem 5.7 by redefining s_7 as $s_7 := \rho_{2\to 5}(t) \oplus s_6$. Then s_{12} yields the 5-expression s that generates $\mathcal{G}$. Note that in s, the $\eta_{3,5}$ operation applied to s_7 introduces not only the edge $\{V_2, V_8\}$ but also $\{V_2, V_6\}$. Likewise, by the $\eta_{3,5}$ operation applied to s_{11}, the edge $\{V_5, V_6\}$ is introduced.

By the tractability results recalled in section 2.2, Theorem 5.7 immediately yields the following.

COROLLARY 5.8 (MS_1 queries and generalized tree-width). *The evaluation of an arbitrary fixed MS_1 query over a class $\mathcal{C}$ of graphs is tractable if $\mathcal{C}$ is of bounded generalized tree-width.*

By Theorem 5.7, bounded generalized tree-width implies bounded clique-width, while the converse is clearly not true. Just consider the class of cliques, whose generalized tree-width is unbounded while the clique-width of all these graphs is 2. Hence, the concept of bounded generalized tree-width does not allow us to push the tractability barrier for the evaluation of MS_1 queries any further. However, the advantage of this new concept is that, in contrast to bounded clique-width, it can be efficiently recognized.

6. Conclusion. In this paper, we have compared query-width, hypertree-width, and several notions of acyclicity of hypergraphs with clique-width. Note that we have mainly considered the clique-width of the *incidence graph* here. When considering restrictions on conjunctive queries, this choice is somehow justified by the fact that the clique-width of the *primal graph* is irrelevant for the tractability. In particular, as we have pointed out in section 2, there are NP-hard classes of queries whose primal graphs are of bounded clique-width. On the other hand, when considering restrictions on the form of the structures, the primal graphs also play an important role. Actually, we have shown that β-acyclicity and bounded clique-width of the primal graph are uncomparable. However, the exact position of bounded clique-width of the primal graph in Figure 1.1 has yet to be determined.

In section 5 we have shown how the insights from the comparison of γ-acyclicity with bounded clique-width can be used for an easy generalization of the tree-width. As long as no polynomial time algorithm for recognizing graphs with clique-width $\leq k$ (for some arbitrary but fixed k) has been found, the search for an appropriate generalization of the tree-width is an interesting research area. We have provided a first and very simple step in this direction, to which further steps should be added; e.g., rather than just considering two-element modules, we might take arbitrary modules and investigate recursively the generalized tree-width of such a module. Likewise, rather than just deleting ear nodes, we might consider the splitting of a graph into its biconnected components. Here we also have the effect that a single biconnected component may contain a module, which was not a module in the overall graph. Likewise, contraction of a module to a single node may allow us to further decompose a graph into biconnected components. The interplay between these two kinds of decompositions deserves further research efforts.

Acknowledgments. We are very grateful to Andreas Brandstädt for pointing out numerous references in the literature. In particular, he suggested the proof of Theorem 4.1, which uses known results and which is thus much simpler than the proof originally provided in this paper. We also thank the anonymous referees, whose detailed comments have greatly helped us to improve the presentation of this work.

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, Reading, MA, 1995.

[2] G. Ausiello, A. D'Atri, and M. Moscarini, *Chordality properties on graphs and minimal conceptual connections in semantic data models*, J. Comput. System Sci., 33 (1986), pp. 179–202.

[3] F. Baader and T. Nipkow, *Term Rewriting and All That*, Cambridge University Press, Cambridge, UK, 1998.

[4] H.-J. Bandelt and H. M. Mulder, *Distance-hereditary graphs*, J. Combin. Theory Ser. B, 41 (1986), pp. 182–208.

[5] W. Bibel, *Constraint satisfaction from a deductive viewpoint*, Artificial Intelligence, 35 (1988), pp. 401–413.

[6] H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.

[7] H. L. Bodlaender, *Treewidth: Algorithmic techniques and results*, in Mathematical Foundations of Computer Science, I. Prívara and P. Ruzicka, eds., Lecture Notes in Comput. Sci. 1295, Springer-Verlag, Berlin, 1997, pp. 19–36.

[8] A. Brandstädt, *personal communication*, 2002.

[9] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A Survey*, SIAM Monogr. Discrete Math. Appl. 3, SIAM, Philadelphia, 1999.

[10] A. Brandstädt and V. V. Lozin, *On the linear structure and clique-width of bipartite permutation graphs*, Ars Combin., 67 (2003), pp. 273–281.

[11] A. K. Chandra and P. M. Merlin, *Optimal implementation of conjunctive queries in relational data bases*, in Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC'77), ACM Press, New York, 1977, pp. 77–90.

[12] C. Chekuri and A. Rajaraman, *Conjunctive query containment revisited*, in Proceedings of the 6th International Conference on Database Theory (ICDT'97), Lecture Notes in Comput. Sci. 1186, Springer-Verlag, Berlin, 1997, pp. 130–144.

[13] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, Cambridge, MA, 1999.

[14] D. G. Corneil, M. Habib, J.-M. Langlinel, and U. R. B. Reed, *Polynomial time recognition of clique-width ≤ 3 graphs, extended abstract*, in Proceedings of the 4th Latin American Symposium on Theoretical Informatics (LATIN 2000), Lecture Notes in Comput. Sci. 1776, Springer-Verlag, Berlin, 2000, pp. 126–134.

[15] B. Courcelle, *Graph rewriting: An algebraic and logic approach*, in Handbook of Theoretical Computer Science, Vol. 2, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 194–241.

[16] B. Courcelle, *Monadic second-order logic of graphs* VII: *Graphs as relational structures*, Theoret. Comput. Sci., 101 (1992), pp. 3–33.

[17] B. Courcelle, J. Engelfriet, and G. Rozenberg, *Handle-rewriting hypergraph grammars*, J. Comput. System Sci., 46 (1993), pp. 218–270.

[18] B. Courcelle, J. A. Makowsky, and U. Rotics, *Linear time solvable optimization problems on graphs of bounded clique-width*, Theory Comput. Syst., 33 (2000), pp. 125–150.

[19] B. Courcelle and S. Olariu, *Upper bounds on the clique-width of graphs*, Discrete Appl. Math., 101 (2000), pp. 77–114.

[20] A. D'Atri and M. Moscarini, *Acyclic Hypergraphs: Their Recognition and Top-Down versus Bottom-Up Generation*, Tech. Report R.29, Consiglio Nazionale delle Ricerche, Istituto di Analisi dei Sistemi ed Informatica, Rome, 1982.

[21] A. D'Atri and M. Moscarini, *Distance-hereditary graphs, Steiner trees, and connected domination*, SIAM J. Comput., 17 (1988), pp. 521–538.

[22] N. Dershowitz and J.-P. Jouannaud, *Rewrite systems*, in Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 243–320.

[23] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999.

[24] R. Fagin, *Degrees of acyclicity for hypergraphs and relational database schemes*, J. ACM, 30 (1983), pp. 514–550.

[25] J. Flum, M. Frick, and M. Grohe, *Query evaluation via tree-decomposition*, J. ACM, 49 (2002), pp. 716–752.

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

[27] M. C. Golumbic and U. Rotics, *On the clique-width of perfect graph classes*, Internat. J.

Found. Comput. Sci., 11 (2000), pp. 423–443.
[28] G. Gottlob, N. Leone, and F. Scarcello, *A comparison of structural CSP decomposition methods*, Artificial Intelligence, 124 (2000), pp. 243–282.
[29] G. Gottlob, N. Leone, and F. Scarcello, *Hypertree decompositions and tractable queries*, J. Comput. System Sci., 64 (2002), pp. 579–627.
[30] G. Gottlob and R. Pichler, *Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width*, in Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001), F. Orejas, P. G. Spirakis, and J. van Leeuwen, eds., Lecture Notes in Comput. Sci. 2076, Springer-Verlag, Berlin, 2001, pp. 708–719.
[31] M. Grohe, T. Schwentick, and L. Segoufin, *When is the evaluation of conjunctive queries tractable?*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC'01), ACM Press, New York, 2001, pp. 657–666.
[32] P. L. Hammer and F. Maffray, *Completely separable graphs*, Discrete Appl. Math., 27 (1990), pp. 85–99.
[33] P. G. Kolaitis and M. Y. Vardi, *Conjunctive-query containment and constraint satisfaction*, in Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposum on Principles of Database Systems (PODS'98), ACM Press, New York, 1998, pp. 205–213.
[34] K. Kunen, *Answer sets and negation as failure*, in Proceedings of the 4th International Conference on Logic Programming (ICLP'87), J.-L. Lassez, ed., MIT Press, Cambridge, MA, 1987, pp. 219–228.
[35] L. J. Stockmeyer, *The Complexity of Decision Problems in Automata Theory*, Ph.D. thesis, Department of Electrical Engineering, MIT, Cambridge, MA, 1974.
[36] M. Yannakakis, *Algorithms for acyclic database schemes*, in Proceedings of the International Conference on Very Large Data Bases (VLDB'81), C. Zaniolo and C. Delobel, eds., IEEE Computer Society, Los Alamitos, CA, 1981, pp. 82–94.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 379–392

THE MINIMUM ALL-ONES PROBLEM FOR TREES*

WILLIAM Y. C. CHEN†, XUELIANG LI†, CHAO WANG†, AND XIAOYAN ZHANG†

Abstract. The minimum all-ones problem was shown to be NP-complete for general graphs. Therefore, it becomes an interesting problem to identify special classes of graphs for which one can find polynomial time algorithms. In this paper we consider this problem for trees. First, for any solution to the all-ones problem for a tree, we give a characterization of the elements in the solution by introducing the concept of the quasi all-ones problem. Then we give the enumeration for the number of solutions in a tree. By using the minimum odd (even) sum problem as subprocess, we obtain a linear time algorithm for the minimum all-ones problem for trees. We also get a linear time algorithm for finding solutions to the all-ones problem in a unicyclic graph.

Key words. lamp lighting problem, all-ones problem, graph algorithm, time complexity

AMS subject classifications. 05C85, 05C70, 90C27, 68Q25, 68R10

DOI. 10.1137/S0097539703421620

1. Introduction. The term *all-ones problem* was introduced by Sutner [9]. The problem has applications in linear cellular automata (see [10] and the references therein) and is cited as follows: Suppose each square of an $n \times n$ chessboard is equipped with an indicator light and a button. If the button of a square is pressed, the light of that square will change from off to on, and vice versa; the same happens to the lights of all the edge-adjacent squares. Initially all lights are off. Now, consider the following questions: is it possible to press a sequence of buttons in such a way that in the end all lights are on? This is referred to as the *all-ones problem*. If there is such a solution, how can we find it? And finally, how can we find a solution that presses as few buttons as possible? This is referred to as the *minimum all-ones problem*. All the above questions can be asked for arbitrary graphs. Here and in what follows, we consider connected simple undirected graphs only. One can deal with disconnected graphs component by component. For all terminology and notation on graphs, we refer to [6]. An equivalent version of the all-ones problem was proposed by Peled in [7], where it was called the *lamp lighting problem*. The rule of the all-ones problem is called the σ^+ rule on graphs, which means that a button lights not only its neighbors but also its own light. If a button lights only its neighbors but not its own light, this rule on graphs is called the σ rule.

The all-ones problem has been extensively studied recently; see Sutner [11, 12], Barua and Ramakrishnan [1], and Dodis and Winkler [2]. Using linear algebra, Sutner [10] proved that it is always possible to light every lamp in any graph by the σ^+ rule. Lossers [5] gave another beautiful proof also by using linear algebra. A graph-theoretic proof was given by Erikisson, Eriksson, and Sjöstrand [3]. So, the existence of solutions of the all-ones problem for general graphs was solved already. Galvin [4] gave a graph-theoretic algorithm of linear time to find solutions for trees. In [8], Sutner proved that the minimum all-ones problem is NP-complete in general. Therefore, it becomes an interesting question to identify special classes of graphs for

*Received by the editors January 25, 2003; accepted for publication (in revised form) November 3, 2003; published electronically February 18, 2004. This work was done under the auspices of the "973" Project on Mathematical Mechanization and the NSFC.
http://www.siam.org/journals/sicomp/33-2/42162.html
†Center for Combinatorics and LPMC, Nankai University, Tianjin 300071, People's Republic of China (x.li@eyou.com).

which one can find polynomial time algorithms. It is the main result of this paper that there exists a linear time algorithm for the minimum all-ones problem for trees.

In graph-theoretic terminology, a solution to the all-ones problem with σ^+-rule can be stated as follows: Given a graph $G = (V, E)$, where V and E denote the node-set and the edge-set of G, respectively, a subset X of V is a solution if and only if for every node v of G the number of nodes in X adjacent to or equal to v is odd. Such a subset X is called an *odd parity cover* in [10]. So, the all-ones problem can be formulated as follows: Given a graph $G = (V, E)$, does a subset X of V exist such that for all nodes $v \in V - X$, the number of nodes in X adjacent to v is odd, while for all nodes $v \in X$, the number of nodes in X adjacent to v is even? If there exists a solution, how can one find it with minimum cardinality?

This paper is organized as follows. In section 2, we give, for any solution to the all-ones problem for a tree, a characterization of the elements in the solution by introducing the concept of the quasi all-ones problem. This leads to an enumeration for the number of solutions in a tree. In section 3, we give a linear time algorithm to the minimum all-ones problem for trees. In the concluding section, section 4, we give a linear time algorithm for constructing solutions to the all-ones problem in a unicyclic graph. An open problem on the all-colors problem is also proposed, generalizing the concept of the all-ones problem.

2. Characterization and enumeration of solutions for trees. It is easy to see that if a given graph G can be partitioned into two disjoint subgraphs G_1 and G_2 such that G_1 is Eulerian and every node of G_2 is adjacent to an odd number of nodes of G_1, then by pressing all the buttons on the nodes of G_1 all the lights will be on, and vice versa. However, it is very difficult to find an Eulerian subgraph with such a property in a large graph G. Sutner [9] posed the question of whether there is a graph-theoretic method to find a solution for the all-ones problem for trees. Galvin [4] solved this question in the following way: Consider a rooted tree, drawn like a family tree, with the root at the top. The nodes will be divided into 3 classes: outcasts, oddballs, and rebels. The classification is defined inductively, from the bottom up, as follows:

- All of the childless nodes or leaves are rebels.
- A node, other than a leaf, is called a *rebel* if it has no oddball children and an even number of its children are rebels.
- A node is called an *oddball* if it has no oddball children and an odd number of its children are rebels.
- A node is called an *outcast* if at least one of its children is an oddball.

We sometimes simply call a node *r-type, b-type, or o-type* if it belongs to the rebel class, the oddball class, or the outcast class, respectively. For examples, see Figure 1.

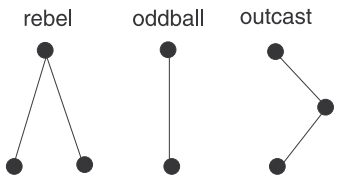


FIG. 1. *The roots of the rooted trees are r-type, b-type, and o-type.*

GALVIN ALGORITHM [4]. Membership in a solution C is defined inductively from the top down. An outcast is excluded from the membership in C. A rebel will be

a member of C if and only if its parent is not a member of C; in particular, if the parentless node (the root) is a rebel, then it will be a member of C. The oddballs will join C in whatever numbers are needed to make their parents' closed neighborhood contain an odd number of members. It is easy to check that C is a solution.

In this section, from the idea of the Galvin algorithm we determine whether or not the root of a tree is in a solution of the all-ones problem and the quasi all-ones problem, defined next. This will be used in enumerating the number of solutions for the all-ones problem. This will also be used in solving the minimum all-ones problem for trees and in constructing solutions to the all-ones problem for unicyclic graphs.

DEFINITION 2.1. *For a rooted tree, the quasi all-ones problem is to find a subset C of nodes such that for every node v except for the root of the tree, the number of nodes in C adjacent to v or equal to v is odd, while for the root, the number is even. C is called a solution to the quasi all-ones problem.*

THEOREM 2.1. *For a rooted tree,*

(1) *if the root is a rebel, then*
   (1.1) *the all-ones problem has a solution if and only if the root belongs to the solution;*
   (1.2) *the quasi all-ones problem has a solution if and only if the root does not belong to the solution;*
(2) *if the root is an oddball, then*
   (2.1) *the all-ones problem has a solution no matter whether the root belongs to the solution;*
   (2.2) *the quasi all-ones problem does not have any solution no matter whether the root belongs to the solution;*
(3) *if the root is an outcast, then*
   (3.1) *the all-ones problem has a solution if and only if the root does not belong to the solution;*
   (3.2) *the quasi all-ones problem has a solution if and only if the root does not belong to the solution.*

Proof. We prove this theorem by induction on the depth s of the rooted tree, which is defined as the maximal distance from root to leaves. In particular, a rooted tree with only one node is of depth 0. It is easy to see that if the root of a tree is a rebel, then the depth can be any nonnegative integer. For a tree with an oddball root, the depth will be at least 1, while for a tree with an outcast root, the depth will be at least 2.

If a rooted tree is of depth 0, then the root must be a rebel node. Any solution to the all-ones problem must contain the root and any solution to the quasi all-ones problem does not contain the root, which means that (1.1) and (1.2) hold when the depth is 0. It is also easy to check that (2.1) and (2.2) hold when the (least possible) depth is 1, and (3.1) and (3.2) hold when the (least possible) depth is 2.

Next, suppose that for any rooted tree whose depth is less than s, all the statements of the theorem are true. Then, for a rooted tree whose depth is s, we distinguish three cases.

Case 1. *The root is a rebel.* Assume that the children of the root are $t_1^{(r)}, \ldots, t_{2k}^{(r)}$, $t_{2k+1}^{(o)}, \ldots, t_m^{(o)}$, where $k \geq 0$, the subtree rooted at $t_i^{(r)}$ $(1 \leq i \leq 2k)$, denoted by $T_i^{(r)}$, is a subtree with a rebel root and with depth less than s, and the subtree rooted at $t_j^{(o)}$ $(2k+1 \leq j \leq m)$, denoted by $T_j^{(o)}$, is a subtree with an outcast root and with depth less than s. Then, from the induction hypothesis, for an outcast-rooted tree the all-ones problem or quasi all-ones problem has a solution if and only if the outcast

root does not belong to the solution. So we can ignore the case for outcast-rooted subtree.

Suppose that the all-ones problem for the rebel-rooted tree has a solution, denoted by C. If the root $r \notin C$, then $C(T_i^{(r)})$ $(1 \le i \le 2k)$ (which is the restriction of C on the subtree $T_i^{(r)}$) is the solution to the all-ones problem for the rebel-rooted subtree $T_i^{(r)}$ with depth less than s. From the induction hypothesis, $t_i^{(r)} \in C(T_i^{(r)})$, and so $t_i^{(r)} \in C$. However, we know that the number of rebel children of the rebel root is even. So, the root cannot be covered odd times by C, and hence C is not an odd parity cover, a contradiction. Thus, if the all-ones problem for a rebel-rooted tree with depth s has a solution, then the root must belong to the solution.

Conversely, consider each rebel-rooted subtree $T_i^{(r)}$ $(1 \le i \le 2k)$ and each outcast-rooted subtree $T_j^{(o)}$ $(2k+1 \le j \le m)$, whose root $t_i^{(r)}, t_j^{(o)}$ is a rebel child and an outcast child of the root r, respectively. Then the quasi all-ones problem for each of them has a solution, denoted by $C(T_i^{(r)}), C(T_j^{(o)})$, respectively. Note that

$$t_i^{(r)} \notin C(T_i^{(r)}) \ (1 \le i \le 2k), \qquad t_j^{(o)} \notin C(T_j^{(o)}) \ (2k+1 \le j \le m).$$

It is easy to check that $C = \{r\} \bigcup (\bigcup_{i=1}^{2k} C(T_i^{(r)})) \bigcup (\bigcup_{j=2k+1}^{m} C(T_j^{(o)}))$ is a solution to the all-ones problem for the original rebel-rooted tree with depth s. So (1.1) holds when the depth is s.

Similarly we can prove (1.2).

Case 2. *The root is an oddball.* Assume that the children of the root are $t_1^{(r)}, \ldots, t_{2k-1}^{(r)}, t_{2k}^{(o)}, \ldots, t_m^{(o)}$, where $k \ge 1$, the subtree rooted at $t_i^{(r)}$ $(1 \le i \le 2k-1)$, denoted by $T_i^{(r)}$, is a rebel-rooted subtree with depth less than s, and the subtree rooted at $t_j^{(o)}$ $(2k \le j \le m)$, denoted by $T_j^{(o)}$, is an outcast-rooted subtree with depth less than s. Similar to the above discussion, we can ignore the case for outcast-rooted subtree.

Suppose that the quasi all-ones problem for the oddball-rooted tree has a solution, denoted by C. If the root $r \in C$ (or $r \notin C$), then $C(T_i^{(r)})$ $(1 \le i \le 2k-1)$ is a solution to the quasi all-ones problem (or the all-ones problem) for the rebel-rooted subtree $T_i^{(r)}$ with depth less than s. From the induction hypothesis, we have that $t_i^{(r)} \notin C(T_i^{(r)})$ (or $t_i^{(r)} \in C(T_i^{(r)})$), and so $t_i^{(r)} \notin C$ (or $t_i^{(r)} \in C$). However, we know that the number of rebel children of the oddball root is odd. So, the root is covered odd times by C, which means that C is a solution to the all-ones problem for the original oddball-rooted tree, a contradiction. Thus (2.2) holds when the depth is s.

Next, we prove (2.1). Consider each rebel-rooted subtree $T_i^{(r)}$ $(1 \le i \le 2k-1)$ and each outcast-rooted subtree $T_j^{(o)}$ $(2k \le j \le m)$, whose root $t_i^{(r)}, t_j^{(o)}$ is a rebel child and an outcast child of the root r, respectively. We discuss the following two cases.

First, the quasi all-ones problem for each of them has a solution, denoted by $C(T_i^{(r)}), C(T_j^{(o)})$, respectively. Note that

$$t_i^{(r)} \notin C(T_i^{(r)}) \ (1 \le i \le 2k-1), \qquad t_j^{(o)} \notin C(T_j^{(o)}) \ (2k \le j \le m).$$

It is easy to check that $C = \{r\} \bigcup (\bigcup_{i=1}^{2k-1} C(T_i^{(r)})) \bigcup (\bigcup_{j=2k}^{m} C(T_j^{(o)}))$ is a solution to the all-ones problem for the original oddball-rooted tree with depth s. Here the root belongs to the solution.

Second, the all-ones problem for each rebel-rooted subtree $T_i^{(r)}$ $(1 \le i \le 2k-1)$ has a solution, denoted by $C(T_i^{(r)})$. Then the all-ones problem for each outcast-rooted subtree $T_j^{(o)}$ $(2k \le j \le m)$ has a solution, denoted by $C(T_j^{(o)})$. Note that

$$t_i^{(r)} \in C(T_i^{(r)}) \ (1 \le i \le 2k-1), \qquad t_j^{(o)} \notin C(T_j^{(o)}) \ (2k \le j \le m).$$

It is easy to check that $C = (\bigcup_{i=1}^{2k-1} C(T_i^{(r)})) \bigcup (\bigcup_{j=2k}^{m} C(T_j^{(o)}))$ is a solution to the all-ones problem for the original oddball-rooted tree with depth s. Here the root does not belong to the solution. Thus (2.1) holds when the depth is s.

Case 3. *The root is an outcast.* Assume that the children of the root are $t_1^{(b)}, \ldots, t_k^{(b)}, t_{k+1}^{(r)}, \ldots, t_l^{(r)}, t_{l+1}^{(o)}, \ldots, t_m^{(o)}$, where $k \ge 1$, the subtree rooted at $t_i^{(b)}$ $(1 \le i \le k)$, denoted by $T_i^{(b)}$, is an oddball-rooted subtree with depth less than s, the subtree rooted at $t_i^{(r)}$ $(k+1 \le i \le l)$, denoted by $T_i^{(r)}$, is a rebel-rooted subtree with depth less than s, and the subtree rooted at $t_i^{(o)}$ $(l+1 \le i \le m)$, denoted by $T_i^{(o)}$, is an outcast-rooted subtree with depth less than s. A similar argument can cover the proof of this case. □

From the theorem, we have the following remarks.

Remark 2.1.

1. For any solution to the all-ones problem for a rooted tree,
   (a) if the root is a rebel, it must belong to the solution;
   (b) if the root is an outcast, it cannot belong to the solution;
   (c) if the root is an oddball, both cases are possible; i.e., it may or may not belong to the solution.
2. If there exists a solution to the quasi all-ones problem for a rooted tree, then the root cannot be an oddball and
   (a) if the root is a rebel, then it cannot belong to the solution;
   (b) if the root is an outcast, then it cannot belong to the solution.

Remark 2.2. If there exists a solution to the all-ones problem or the quasi all-ones problem for a rooted tree, then

1. if the root is a rebel or an oddball, both cases are possible; i.e., it may or may not belong to the solution;
2. if the root is an outcast, it cannot belong to the solution.

From the above clear analysis, we can get the following enumeration result.

THEOREM 2.2. *If a rooted tree has p oddball nodes and q outcast nodes, then the number of solutions to the all-ones problem for the tree is 2^{p-q}.*

Proof. From Theorem 2.1 and Remarks 2.1 and 2.2, we can deduce the following facts: First, for a rebel node v, if its parent node is not contained in a solution C, then v is contained in C, whereas if its parent node is contained in C, then v is not contained in C. Second, the outcast nodes cannot be contained in any solution.

If the root of the tree is a rebel or an outcast node, then every outcast node needs one of its oddball children to match its rebel children so that the outcast node will be lighted without itself in the solution. So, at each outcast node, its oddball children have degrees of freedom equal to the number of oddball children minus 1. Therefore, the number of solutions to the all-ones problem for a tree is exactly 2^{p-q}. If the root of the tree is an oddball, since the root can have two choices, i.e., in a solution, or not, we have that the number of solutions is $2(2^{p-1-q}) = 2^{p-q}$. The proof is complete. □

From the results so far, we can say that the all-ones problem for trees has a satisfactory solution. It is natural to ask about the minimum all-ones problem for

trees. The minimum all-ones problem is NP-complete for general graphs [8]. However, it can be solved easily for some special classes of graphs; for example, a path with n nodes has an optimal solution with $\lceil \frac{n}{3} \rceil$ nodes, and a cycle with n nodes has an optimal solution with $\frac{n}{3}$ nodes if $n = 0 \mod 3$, and n nodes otherwise. For an arbitrary tree, no such succinct formula has been known for the number of nodes in an optimal solution. However, we can ask whether there is a polynomial time algorithm for trees. At first look, we cannot see if there is such an algorithm, because from our Theorem 2.2 we know that the number of solutions could be exponentially large. However, we do obtain a polynomial time algorithm for trees by using the characterization in Theorem 2.1 and Remarks 2.1 and 2.2. Actually, what we get is a linear time algorithm.

3. The minimum all-ones problem for trees. In order to give our algorithm, we need to introduce a new problem, called the *minimum odd (even) sum problem*, which is described as the following linear program.

For the matrix $M_{2\times n} = (m_{ij})_{2\times n}$, $i \in \{0,1\}$, $j \in \{1,2,\cdots,n\}$, $m_{ij} \in Z^+$, the *minimum odd sum problem* is defined as

$$\min \sum_{j=1}^{n} m_{0j}x_{0j} + m_{1j}x_{1j},$$

$$\begin{cases} \sum_{j=1}^{n} x_{1j} = 1 \mod 2 \\ x_{0j} + x_{1j} = 1, \quad j = 1,2,\ldots,n \\ x_{ij} \in \{0,1\}, \quad i \in \{0,1\}. \end{cases}$$

Note that $m_{0j}x_{0j} + m_{1j}x_{1j}$ is equal to m_{1j} if $x_{1j} = 1$, or m_{0j} if $x_{1j} = 0$. So $m_{0j}x_{0j} + m_{1j}x_{1j}$ can be written as $m_{x_{1j}j}$. For convenience, we replace x_{1j} by y_j. Then it is easy to see that the above linear program is equivalent to the following one:

$$\min \sum_{j=1}^{n} m_{y_j j},$$

$$\begin{cases} \sum_{j=1}^{n} y_j = 1 \mod 2 \\ y_j \in \{0,1\} \end{cases}$$

Algorithm for the minimum odd sum problem.

Input. A matrix $M_{2\times n}$.

Step 1. Choose a minimum element from every column of $M_{2\times n}$ (if both elements in a column are the same, choose one of the them). Then sum up the first subscripts of all the chosen elements, denoted by S. If $S = 1 \mod 2$, go to Step 3; otherwise, go to Step 2;

Step 2. Calculate the absolute value of the difference of the two elements in every column. Choose one of the columns with the minimum absolute values. In this column, we choose the hitherto unselected element and forget about the chosen element, then go to Step 3;

Step 3. Sum up all the chosen elements, which gives the optimal value $\min \sum_{j=1}^{n} m_{y_j j}$.

Theorem 3.1. *The above algorithm correctly solves the minimum odd sum problem, and the time complexity is linear.*

Proof. The first statement of the theorem is proved as follows. Since the minimum odd sum problem asks for a unique element from every column, our greedy algorithm

picks up the minimum element from every column. If the sum of the first subscripts of all the chosen elements satisfies that $\sum_{j=1}^{n} y_j = 1 \bmod 2$, then the sum of all the chosen elements is exactly the optimal value $\min \sum_{j=1}^{n} m_{y_j j}$. If $\sum_{j=1}^{n} y_j \neq 1 \bmod 2$, we only need to adjust the elements slightly so that the sum of the first subscripts of all the chosen elements satisfies that $\sum_{j=1}^{n} y_j = 1 \bmod 2$. Because we adjust elements in the column where the minimum absolute value of the difference of the two elements is attained from Step 2, it is easy to see that the chosen elements after adjusting have the minimum sum among the feasible solutions to the odd sum problem; i.e., the chosen elements consist of an optimal solution.

For the second statement of the theorem, since every step uses linear time, the total time is $O(n)$. The proof is complete. □

From the above discussion, we can enumerate the number of optimal solutions to the minimum odd sum problem. Suppose that the absolute value of the difference of the two elements in the ith column is $|d_i|$.

If $\min\{|d_i| \mid i = 1, 2, \ldots, n\} = s > 0$, then

1. if the sum of the first subscripts of all the chosen elements satisfies that $\sum_{j=1}^{n} y_j = 1 \bmod 2$ in Step 1, it is straightforward to see that the problem has a unique optimal solution;

2. if $\sum_{j=1}^{n} y_j \neq 1 \bmod 2$, the only possible ways to adjust the chosen elements have to be done in the set of the columns $\{i \mid |d_i| = s,\ i = 1, 2, \ldots, n\}$, say, r such columns in all. Since we can do the adjustment in any one of the r such columns, the problem has r optimal solutions.

If $\min\{|d_i| \mid i = 1, 2, \ldots, n\} = s = 0$, and supposing $|\{i \mid |d_i| = 0,\ i = 1, 2, \ldots, n\}| = r$, then

1. if $\sum_{j=1}^{n} y_j = 1 \bmod 2$, the only possible ways to adjust the chosen elements have to be done in an even number of the columns $\{i \mid |d_i| = 0,\ i = 1, 2, \ldots, n\}$. So, the problem has T_0 optimal solutions, where $T_0 = \binom{r}{0} + \binom{r}{2} + \binom{r}{4} + \cdots = 2^{r-1}$;

2. if $\sum_{j=1}^{n} y_j \neq 1 \bmod 2$, the only possible ways to adjust the chosen elements have to be done in an odd number of the columns $\{i \mid |d_i| = 0,\ i = 1, 2, \ldots, n\}$. So, the problem has T_1 optimal solutions, where $T_1 = \binom{r}{1} + \binom{r}{3} + \binom{r}{5} + \cdots = 2^{r-1}$.

Replacing $\sum_{j=1}^{n} y_j = 1 \bmod 2$ in the minimum odd sum problem by $\sum_{j=1}^{n} y_j = 0 \bmod 2$, we then get a new problem, called the *even sum problem*. It can be solved in the same way as above. The details are omitted.

Now we give our linear time algorithm to the minimum all-ones problem for trees. The algorithm uses induction on the number of layers of a tree and the minimum odd or even sum algorithm as subprocess.

First of all, we give the definition of layers for a rooted tree as follows: The ith *layer* of the tree is composed of the nodes with distance i from the root for $i = 0, 1, 2, \ldots$. Suppose the tree has s layers. Then for any $i < s$, every node except the leaves in the ith layer can be considered the root of a small tree with depth 1, which is simply called a *small tree* in what follows. We divide the small trees into the following three types.

Type I. A type I small tree has an r-type root. For such a small tree, we can assume that the children of its root are $t_1^{(r)}, \ldots, t_{2k}^{(r)}, t_{2k+1}^{(o)}, \ldots, t_m^{(o)}$, where $k \geq 0$, the subtree rooted at $t_i^{(r)}$ $(1 \leq i \leq 2k)$ is denoted by $T_i^{(r)}$, and the subtree rooted at $t_j^{(o)}$ $(2k + 1 \leq j \leq m)$ is denoted by $T_j^{(o)}$. An example of type I small trees is shown in Figure 2(a).

Type II. A type II small tree has a b-type root. For such a small tree, we can

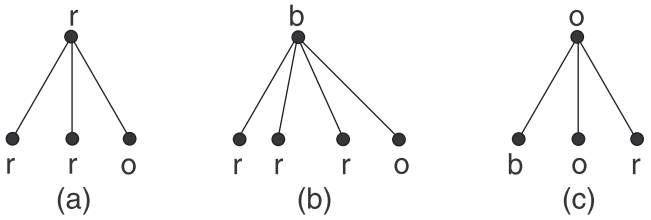


FIG. 2. *Examples of small trees of types* I, II, *and* III.

assume that the children of its root are $t_1^{(r)}, \ldots, t_{2k-1}^{(r)}, t_{2k}^{(o)}, \ldots, t_m^{(o)}$, where $k \geq 1$, the subtree rooted at $t_i^{(r)}$ $(1 \leq i \leq 2k-1)$ is denoted by $T_i^{(r)}$, and the subtree rooted at $t_j^{(o)}$ $(2k \leq j \leq m)$ is denoted by $T_j^{(o)}$. An example of type II small trees is shown in Figure 2(b).

Type III. A type III small tree has an o-type root. For such a small tree, we can assume that the children of its root are $t_1^{(b)}, \ldots, t_k^{(b)}, t_{k+1}^{(r)}, \ldots, t_l^{(r)}, t_{l+1}^{(o)}, \ldots, t_m^{(o)}$, where $k \geq 1$, the subtree rooted at $t_i^{(b)}$ $(1 \leq i \leq k)$ is denoted by $T_i^{(b)}$, the subtree rooted at $t_i^{(r)}$ $(k+1 \leq i \leq l)$ is denoted by $T_i^{(r)}$, and the subtree rooted at $t_i^{(o)}$ $(l+1 \leq i \leq m)$ is denoted by $T_i^{(o)}$. An example of type III small trees is shown in Figure 2(c).

By executing our algorithm layer by layer, from the bottom up, we are going to tag each node v in the present layer with a pair of sets of nodes. Then we can get an optimal solution in linear time. If v is a leaf, it is tagged by Step 0 of our algorithm; if not, its tagged pair of sets can be obtained from the following three cases.

Case 1. For every r-type leaf t_i^r of the small trees rooted at v, since the leaf is in the previous layer, we have an optimal solution $C_1(T_i^{(r)})$ to the all-ones problem for the subtree rooted at $t_i^{(r)}$ and an optimal solution $C_2(T_i^{(r)})$ to the quasi all-ones problem for the same subtree.

Case 2. For every b-type leaf t_i^b of the small trees rooted at v, we have an optimal solution $C_1(T_i^{(b)})$ to the all-ones problem for the subtree rooted at t_i^b such that the root of the subtree belongs to $C_1(T_i^{(b)})$ and an optimal solution $C_2(T_i^{(b)})$ to the all-ones problem for this subtree such that the root of the subtree does not belong to $C_2(T_i^{(b)})$.

Case 3. For every o-type leaf t_i^o of the small trees rooted at v, we have an optimal solution $C_1(T_i^{(o)})$ to the all-ones problem for the subtree rooted at t_i^o and an optimal solution $C_2(T_i^{(o)})$ to the quasi all-ones problem for the subtree.

Note that the above pair for every leaf of a tree is clearly determined at the beginning of our algorithm.

ALGORITHM FOR THE MINIMUM ALL-ONES PROBLEM FOR TREES.

Input. A rooted tree T with s layers, and a pair $\{C_1(v), C_2(v)\}$ of sets for each node v of the tree.

Step 0. Initially, for every leaf $t^{(r)}$ of T, set $\{\{t^{(r)}\}, \emptyset\}$, which means that $\{t^{(r)}\}$ is the optimal solution to the all-ones problem for the subtree with the single node $t^{(r)}$, and $\emptyset$ is the optimal solution to the quasi all-ones problem for the single node subtree.

Step 1. Inductively generate the pair for every node of T layer by layer, from the bottom up, till we arrive at the root of the tree. Suppose that the present layer is the ith layer. If $i \geq 0$, go to Step 2; otherwise, go to Step 4;

Step 2. We distinguish the following three cases to generate the pairs. For every small tree rooted in the ith layer, the algorithm works as follows:

1. The tree is type I. Denote its root by r^*. Suppose that the children of r^* are $t_1^{(r)}, \ldots, t_{2k}^{(r)}, t_{2k+1}^{(o)}, \ldots, t_m^{(o)}$, where $k \geq 0$. We already knew that $C_1(t_i^{(r)}) = C_1(T_i^{(r)})$ and $C_2(t_i^{(r)}) = C_2(T_i^{(r)})$ for every r-type leaf as a root for the subtree $T_i^{(r)}$, where $1 \leq i \leq 2k$, and $C_1(t_i^{(o)}) = C_1(T_i^{(o)})$ and $C_2(t_i^{(o)}) = C_2(T_i^{(o)})$ for every o-type leaf as a root for the subtree $T_i^{(o)}$, where $2k+1 \leq i \leq m$. Then set
$$C_1(r^*) = \{r^*\} \bigcup \left(\bigcup_{i=1}^{2k} C_2(t_i^{(r)}) \right) \bigcup \left(\bigcup_{j=2k+1}^{m} C_2(t_j^{(o)}) \right)$$
as the optimal solution to the all-ones problem of the subtree rooted at r^*, and set
$$C_2(r^*) = \left(\bigcup_{i=1}^{2k} C_1(t_i^{(r)}) \right) \bigcup \left(\bigcup_{j=2k+1}^{m} C_1(t_j^{(o)}) \right)$$
as the optimal solution to the quasi all-ones problem of the subtree rooted at r^*.
2. The tree is type II. Denote its root by b^*. Suppose that the children of b^* are $t_1^{(r)}, \ldots, t_{2k-1}^{(r)}, t_{2k}^{(o)}, \ldots, t_m^{(o)}$, where $k \geq 1$. We already knew that $C_1(t_i^{(r)}) = C_1(T_i^{(r)})$ and $C_2(t_i^{(r)}) = C_2(T_i^{(r)})$ for every r-type leaf as a root for the subtree $T_i^{(r)}$, where $1 \leq i \leq 2k-1$, and $C_1(t_i^{(o)}) = C_1(T_i^{(o)})$ and $C_2(t_i^{(o)}) = C_2(T_i^{(o)})$ for every o-type leaf as a root for the subtree $T_i^{(o)}$, where $2k-1 \leq i \leq m$. Then set
$$C_1(b^*) = \{b^*\} \bigcup \left(\bigcup_{i=1}^{2k-1} C_2(t_i^{(r)}) \right) \bigcup \left(\bigcup_{j=2k}^{m} C_2(t_j^{(o)}) \right)$$
as the optimal solution to the all-ones problem of the subtree rooted at b^* such that b^* belongs to the optimal solution, and set
$$C_2(b^*) = \left(\bigcup_{i=1}^{2k-1} C_1(t_i^{(r)}) \right) \bigcup \left(\bigcup_{j=2k}^{m} C_1(t_j^{(o)}) \right)$$
as the optimal solution to the all-ones problem of the subtree rooted at b^* such that b^* does not belong to the optimal solution.
3. The tree is type III. Denote its root by o^*. Suppose that the children of o^* are $t_1^{(b)}, \ldots, t_k^{(b)}, t_{k+1}^{(r)}, \ldots, t_l^{(r)}, t_{l+1}^{(o)}, \ldots, t_m^{(o)}$, where $k \geq 1$. Use the pairs of sets on the nodes $t_1^{(b)}, \ldots, t_k^{(b)}$ to make a two-dimensional matrix $C_{2\times k} = (c_{ij})_{2\times k}$ such that $|C_2(t_i^{(b)})|$ is the value of the element c_{0i} in $(c_{ij})_{2\times k}$ and $|C_1(t_i^{(b)})|$ is the value of the element c_{1i}.

Remark 3.1. From Theorem 2.1 and Remarks 2.1 and 2.2, any solution to the all-ones problem (or the quasi all-ones problem) of the subtree rooted at o^* cannot contain the o-type root o^*. This means that all the r-type children of the o-type

root must be contained in the solution. Because the solution must contain an odd (or even) number of children of the o-type root, we have to employ our minimum odd sum algorithm or the minimum even sum algorithm to choose some of the b-type children into an optimal solution, according to the parity of $l-k$.

Step 2 (cont'd). If $l-k$ is even (odd), we use the minimum odd (even) sum algorithm to choose the elements in $(c_{ij})_{2\times k}$. Suppose that the union of the elements chosen from the 0th row in $(c_{ij})_{2\times k}$ is $\bigcup_{p=1}^{n_1} c_{0j_p}$, and the union of the elements chosen from the first row in $(c_{ij})_{2\times k}$ is $\bigcup_{q=1}^{n_2} c_{1j_q}$, where $n_1+n_2=k$. Then we set

$$C_1(o^*)=\left(\bigcup_{p=1}^{n_1} C_2(t_{j_p}^{(b)})\right)\bigcup\left(\bigcup_{q=1}^{n_2} C_1(t_{j_q}^{(b)})\right)\bigcup\left(\bigcup_{i=k+1}^{l} C_1(t_i^{(r)})\right)\bigcup\left(\bigcup_{i=l+l}^{m} C_1(t_i^{(o)})\right)$$

as the optimal solution to the all-ones problem of the subtrees rooted at o^*. Next, we use the minimum even (odd) sum algorithm to choose the elements in $(c_{ij})_{2\times k}$. Suppose that the union of the elements chosen from the 0th row in $(c_{ij})_{2\times k}$ is $\bigcup_{p=1}^{n_1} c_{0j_p}$, and the union of the elements chosen from the first row in $(c_{ij})_{2\times k}$ is $\bigcup_{q=1}^{n_2} c_{1j_q}$, where $n_1+n_2=k$. Then we set

$$C_2(o^*)=\left(\bigcup_{p=1}^{n_1} C_2(t_{j_p}^{(b)})\right)\bigcup\left(\bigcup_{q=1}^{n_2} C_1(t_{j_q}^{(b)})\right)\bigcup\left(\bigcup_{i=k+1}^{l} C_1(t_i^{(r)})\right)\bigcup\left(\bigcup_{i=l+l}^{m} C_1(t_i^{(o)})\right)$$

as the optimal solution to the quasi all-ones problem of the subtrees rooted at o^*.

Step 3. $i := i-1$, go to Step 1;

Step 4. We are now ready to give an optimal solution for the rooted tree T from the pair on the root by distinguishing the following three cases: (i) If the root is of r-type, denoted by r^*, then the $C_1(r^*)$ of the pair is an optimal solution. (ii) If the root is of b-type, denoted by b^*, then the $C_1(b^*)$ of the pair is a candidate for the optimal solution such that b^* belongs to the candidate solution, and the $C_2(b^*)$ of the pair is another candidate for the optimal solution such that b^* does not belong to the candidate solution. Now, compare the values of $|C_1(b^*)|$ and $|C_2(b^*)|$. Suppose that $|C_t(b^*)| = \min\{|C_1(b^*)|, |C_2(b^*)|\}$, $t\in\{1,2\}$. Then we choose $C_t(b^*)$ as an optimal solution. (iii) If the root is of o-type, denoted by o^*, then the $C_1(o^*)$ of the pair is an optimal solution.

THEOREM 3.2. *The above algorithm outputs an optimal solution to the all-ones problem of a given tree T, and the time complexity is linear.*

Proof. In Step 0, we regard every leaf in the bottom of the tree as a subtree whose unique optimal solution to the all-ones problem and the quasi all-ones problem contains exactly the node itself and nothing, respectively. Then the initial values of all the leaves of the tree can be completely determined. The algorithm now proceeds inductively on the number of layers of the tree. Then from the method for constructing solutions in the proof of Theorem 2.1 and from Remarks 2.1, 2.2, and 3.1, it is easy to conclude that the algorithm ensures that all the leaves v of the small trees of all types I, II, and III in every layer have recorded the right information, i.e., the pairs $\{C_1(v), C_2(v)\}$. From these pairs of sets, we can choose the optimal solution for the given tree according to the type of the root of the tree. The first statement of the theorem is thus proved.

For the second statement of the theorem, it is not hard to see that for every layer, the algorithm uses time linear in the number of nodes in the layer, even though

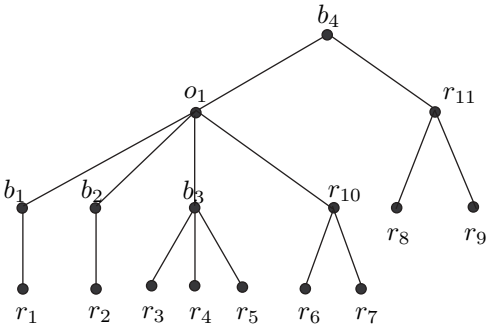


FIG. 3. *Labeling the type for every node.*

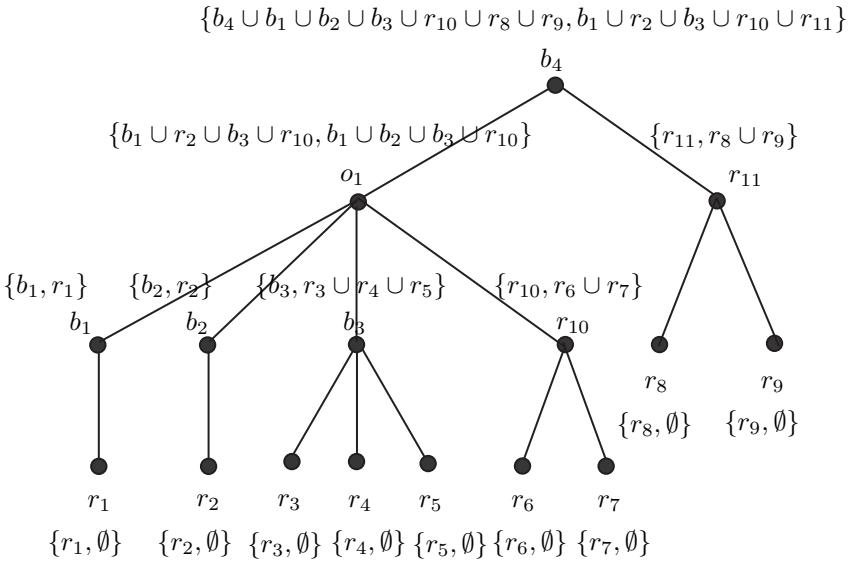


FIG. 4. *An example of our algorithm for the minimum all-ones problem for trees.*

sometimes the minimum odd or even sum algorithm has to be used. Therefore, the total time used by the algorithm is linear for the minimum all-ones problem of the given tree. The proof is complete. □

An example to show our algorithm at work is given in Figures 3 and 4. For every node in Figure 3, the label, by ignoring its subscript, is the type of that node. In Figure 4 we simply use x to denote the set $\{x\}$ with a single element x. Initially, our algorithm sets a pair $\{v, \emptyset\}$ for every leaf v. Then, from the bottom up, the pair for each node in every layer can be generated. Note that the children of o_1 are b_1, b_2, b_3, and r_{10}, and the pairs on the b-type nodes can form a matrix as described in our algorithm: $P_{23} = \left(\begin{smallmatrix} r_1 & r_2 & r_3 \cup r_4 \cup r_5 \\ b_1 & b_2 & b_3 \end{smallmatrix}\right)$. The corresponding numerical matrix is $C_{23} = \left(\begin{smallmatrix} 1 & 1 & 3 \\ 1 & 1 & 1 \end{smallmatrix}\right)$, which will be used in the minimum odd (even) sum algorithm. Note that only r_{10} is an r-type node. Then use the minimum even sum algorithm to get $b_1 \cup r_2 \cup b_3$ (possibly, $r_1 \cup b_2 \cup b_3$) union r_{10} and form $C_1(o_1)$. Use the minimum odd

sum algorithm to get $b_1 \cup b_2 \cup b_3$ (possibly, $r_1 \cup r_2 \cup b_3$) union r_{10} and form $C_2(o_1)$. In the end, by comparing $|C_1(b_4)| = 7$ with $|C_2(b_4)| = 5$, we get an optimal solution $C_2(b_4) = b_1 \cup r_2 \cup b_3 \cup r_{10} \cup r_{11}$. The details about the pair $\{C_1(v), C_2(v)\}$ for every node v are recorded in Figure 4.

4. Concluding remarks. Although the existence of solutions to the all-ones problem for general graphs was proved by linear algebraic methods (see [10, 5]), in [10] Sutner asked whether there is a graph-theoretic proof for the existence. Eriksson, Eriksson, and Sjöstrand [3] gave such a proof. However, how to find a solution efficiently by graph-theoretic algorithms remains unknown. Although, based on the result in [3], one can get a graph-theoretic algorithm inductively, the time complexity is not polynomial, which is upper bounded by $O(n!)$. It is easy to see that for the empty graph with n nodes, their algorithm runs in time $O(n!)$. So, to find a graph-theoretic algorithm of polynomial time for general graphs, some other ideas are needed. For trees, Galvin [4] gave a graph-theoretic algorithm of linear time. In this concluding section, based on the discussion in section 2 we would like to give such an algorithm of linear time for unicyclic graphs.

For convenience, we say that the truth value of a node in G is 1 if it belongs to the solution to the all-ones problem (or the quasi all-ones problem) for G, and 0 otherwise. Recall that a graph G is called *unicyclic* if it contains a unique cycle. In other words, we can regard a unicyclic graph as a cycle attached with each node to a rooted tree, called a *suspended tree*. Note that the depth of a suspended tree can be 0. For simplicity, we say that a node t in the cycle has the same type as the type of the root t of the suspended tree.

Algorithm for unicyclic graphs.

Input. A unicyclic graph G, each node in the unique cycle being labeled by types.

Step 1. If none of the nodes on the cycle is an outcast, then let the truth values of all nodes on the cycle be 1; i.e., take the union of the solutions, each of which is a solution to the all-ones problem for each suspended tree whose root belongs to the solution. Then the union is a solution to the all-ones problem for the whole unicyclic graph.

Step 2. If there are outcast nodes, we fix an order to the nodes on the cycle. Then we cut the cycle by deleting the edge between an outcast node u and the node v before it on the cycle. The unicyclic graph becomes a tree with root v, denoted as T, and the type of every node on the original cycle will be changed as in Figure 5, where the changing of the type of each node is from on the original cycle to on the tree T, and the changing rule is just the same as that in the Galvin method.

By Galvin's algorithm, we can construct a solution X for the tree T. Then we add an edge to connect the node u and the node v in T; then the tree T returns to the original unicyclic graph with a solution X. Because u is an outcast node, no matter whether it is in the unicyclic graph or in the tree, it will not belong to the solution, and hence will not affect the other nodes' truth values, while u's on or off status will probably be affected by v if v belongs to X. If v does not belong to X, then X is a solution to the all-ones problem for the unicyclic graph; otherwise, we only need to change the solution to the all-ones problem into the solution to the quasi all-ones problem, or the other way around for the suspended tree with root u according to the construction method in the proof of Theorem 2.1; then the modified solution is a solution to the all-ones problem for the unicyclic graph. An example is shown in Figure 6.

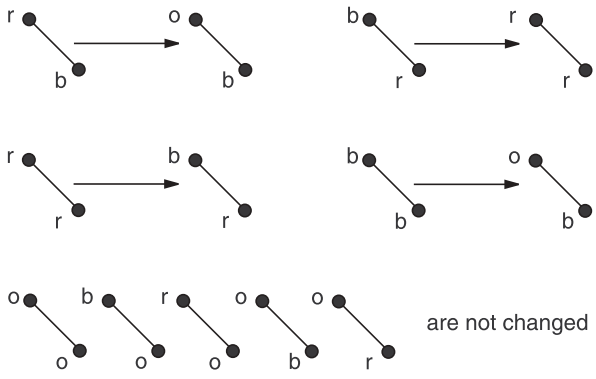


FIG. 5. *Changing of types.*

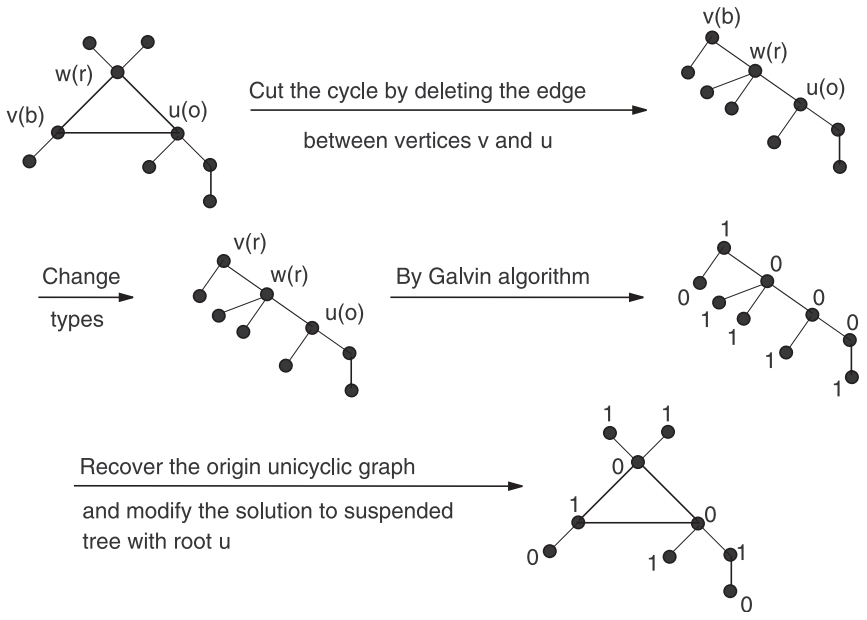


FIG. 6. *An example of our algorithm for unicyclic graphs.*

So, we get the following result.

THEOREM 4.1. *The above algorithm outputs solutions to the all-ones problem for unicyclic graphs, and the time complexity is linear.*

To end this paper, we propose the following problem.

All-colors problem. The so-called *all-colors problem* on graphs is described as follows, which is a natural generalization for the all-ones problem:

For any node of a graph G, it has a color value between 0 and $r-1$. If a node is pressed one time, then the color values of the node and its neighbors are added by 1 under the meaning of modular r. If the initial status is that the color value of every node is 0, then we ask how to press some nodes (maybe many times) to make the color value of every node equal to $r-1$ (or any fixed k such that $1 \leq k \leq r-1$) under

the meaning of modular r. If we ask that the sum of color values of all nodes attains the minimum, the problem is called the *minimum all-colors problem*.

Acknowledgment. The authors are greatly indebted to the referees for their invaluable suggestions and comments, which have substantially improved the presentation of the paper.

REFERENCES

[1] R. Barua and S. Ramakrishnan, *σ-game, σ^+-game and two-dimensional additive cellular automata*, Theoret. Comput. Sci., 154 (1996), pp. 349–366.
[2] Y. Dodis and P. Winkler, *Universal configurations in light-flipping games*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2001, pp. 926–927.
[3] H. Eriksson, K. Eriksson, and J. Sjöstrand, *Note on the lamp lighting problem*, Adv. in Appl. Math., 27 (2001), pp. 357–366.
[4] F. Galvin, *Solution to problem* 88-8, Math. Intelligencer, 11 (2) (1989), pp. 31–32.
[5] O. P. Lossers, *Solution to problem* 10197, Amer. Math. Monthly, 100 (1993), pp. 806–807.
[6] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimizations: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
[7] U. Peled, *Problem* 10197, Amer. Math. Monthly, 99 (1992), p. 162.
[8] K. Sutner, *Additive automata on graphs*, Complex Systems, 2 (1988), pp. 1–28.
[9] K. Sutner, *Problem* 88-8, Math. Intelligencer, 10 (3) (1988), p. 101.
[10] K. Sutner, *Linear cellular automata and the Garden-of-Eden*, Math. Intelligencer, 11 (2) (1989), pp. 49–53.
[11] K. Sutner, *The σ-game and cellular automata*, Amer. Math. Monthly, 97 (1990), pp. 24–34.
[12] K. Sutner, *σ-automata and Chebyshev-polynomials*, Theoret. Comput. Sci., 230 (2000), pp. 49–73.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 393–415

SCHEDULING WITH AND/OR PRECEDENCE CONSTRAINTS*

ROLF H. MÖHRING†, MARTIN SKUTELLA‡, AND FREDERIK STORK§

Abstract. In many scheduling applications it is required that the processing of some job be postponed until some other job, which can be chosen from a pregiven set of alternatives, has been completed. The traditional concept of precedence constraints fails to model such restrictions. Therefore, the concept has been generalized to so-called AND/OR *precedence constraints* which can cope with this kind of requirement. In the context of traditional precedence constraints, feasibility, transitivity, and the computation of earliest start times for jobs are fundamental, well-studied problems. The purpose of this paper is to provide efficient algorithms for these tasks for the more general model of AND/OR precedence constraints. We show that feasibility as well as many questions related to transitivity can be solved by applying essentially the same linear-time algorithm. In order to compute earliest start times we propose two polynomial-time algorithms to cope with different classes of time distances between jobs.

Key words. project scheduling, AND/OR precedence constraints, earliest start schedule, mean payoff games

AMS subject classifications. 90B35, 05C65, 90C27, 91A43, 05C85

DOI. 10.1137/S009753970037727X

1. Introduction.

Definition and motivation of AND/OR precedence constraints. For a given set V of jobs, a precedence constraint comprehends the requirement that a job j cannot be started before another job i has been completed. Precedence constraints are usually given by a set A of ordered pairs (i, j), $i \neq j \in V$, inducing an acyclic digraph $D = (V, A)$ where each node corresponds to a job and each arc represents a precedence constraint. In a feasible implementation of the project, the jobs have to be executed in accordance with the partial order defined by D. Since, in this setting, each job j can only start after the completion of *all* its predecessors in D, we call these precedence constraints AND-*constraints*. However, there are many applications where jobs can be executed as soon as *any* of its predecessors has been completed; we refer to such temporal restrictions as OR-*constraints*. Traditional precedence constraints fail to model this requirement and consequently, the model has been generalized to so-called AND/OR *precedence constraints*. AND/OR precedence constraints can be represented by a set $\mathcal{W}$ of pairs (X, j) with the meaning that job $j \in V$ cannot be executed before some job $i \in X \subseteq (V \setminus \{j\})$ has been completed. We

*Received by the editors August 23, 2000; accepted for publication (in revised form) January 18, 2003; published electronically February 18, 2004. This work was supported in part by the EU Thematic Networks APPOL I & II, Approximation and Online Algorithms (IST-1999-14084 and IST-2001-30012). Part of this research appeared as a two-page abstract [21] in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00).

http://www.siam.org/journals/sicomp/33-2/37727.html

†Technische Universität Berlin, Fakultät II, Institut für Mathematik, Sekr. MA 6-1, Straße des 17. Juni 136, D-10623 Berlin, Germany (moehring@math.tu-berlin.de).

‡Max-Planck Institut für Informatik, Stuhlsatzenhausweg 85, D-66123 Saarbrücken, Germany (skutella@mpi-sb.mpg.de). This author was supported in part by DONET within the frame of the TMR Program (contract ERB FMRX-CT98-0202) while staying at C.O.R.E., Louvain-la-Neuve, Belgium, for the academic year 1998–1999.

§ILOG Deutschland GmbH, Ober-Eschbacher Straße 109, D-61352 Bad Homburg, Germany (fstork@ilog.de). This work was done while this author was at Technische Universität Berlin and received support from Deutsche Forschungsgemeinschaft (DFG), grant Mo 446/3-3.

call such (X, j) pairs *waiting conditions* and job j the *waiting job* for (X, j). Notice that for a singleton $X = \{i\}$, the constraint (X, j) is a traditional AND-constraint (i, j).

An intuitive motivation for AND/OR precedence constraints is noted by Gillies and Liu [12]. An engine head has to be fixed by four bolts. However, one of the bolts may secure the engine head well enough to allow further work on it. If the set X consists of the four jobs to secure the bolts and j represents the further work on the engine head, then the waiting condition (X, j) obviously models the desired temporal dependencies among the jobs. Another motivation is studied by Goldwasser and Motwani [13]. They consider the problem of partially disassembling a given product to reach a single part (or component). In order to remove a certain part, one previously may have to remove other parts which can be modeled by traditional (AND) precedence constraints. However, one may choose to remove that same part of the product from another geometric direction, in which case some other parts must be removed previously. This freedom of choice can be modeled by AND/OR precedence constraints. A third motivation is given by Dinic [6], who considers the setup of new technologies and products. In his model, a new technology requires certain products; on the other hand, a new product can be obtained as an output of one of several new technologies. The latter requirement leads to an OR-constraint, while the first requirement is a classical AND-constraint.

Our research is motivated by problems occurring in resource-constrained project scheduling. Resource constraints can be represented by so-called *minimal forbidden sets*, i.e., inclusion-minimal sets of jobs that cannot be scheduled simultaneously. In order to resolve the resource conflict that occurs due to a minimal forbidden set F, one may choose a job $j \in F$ to be a waiting job for $F \setminus \{j\}$ and introduce the corresponding waiting condition $(F \setminus \{j\}, j)$. Thus, we are able to represent solutions of project scheduling problems by a set $\mathcal{W}$ of waiting conditions. For details we refer to [26].

Connections to other fields and related work. The combinatorial structure of a system $\mathcal{W}$ of waiting conditions occurs in different fields of discrete mathematics and theoretical computer science. In the context of *directed hypergraphs* each $(X, j) \in \mathcal{W}$ represents a hyperarc with a set X of source nodes and a single target node j. Ausiello, d'Atri, and Saccà [3] (see also [4]) generalize transitive closure and reduction algorithms from directed graphs to directed hypergraphs. Another related class of combinatorial objects are *antimatroids* (special greedoids) which can be defined via a set of waiting conditions; see, e.g., [17, page 22]. Furthermore, many problems stemming from artificial intelligence can be formulated by hierarchies of subproblems where different alternatives exist to solve these subproblems; see, e.g., [23]. There, a graphical representation of such hierarchies is called an AND/OR graph.

In the context of scheduling, Goldwasser and Motwani [13] derive inapproximability results for two single-machine scheduling problems with AND/OR precedence constraints. Gillies and Liu [12] consider single- and parallel-machine scheduling problems with different structures of AND/OR precedence constraints; they prove NP-completeness of finding feasible schedules in some settings that are polynomially solvable with traditional precedence constraints. Moreover, they give approximation algorithms for some makespan minimization problems.

A basic, important task in scheduling applications is the computation of earliest start times of jobs. The research on this topic will be discussed in more detail in section 7.1.

Contribution of this paper. For AND-constraints, fundamental problems such as deciding feasibility, finding transitive AND-constraints, and computing earliest start times of jobs can be solved efficiently by applying simple well-known graph algorithms. Most important for the algorithmic treatment is the fact that AND-constraints define acyclic structures on the set V of jobs such that many problems can be solved by considering jobs in the order of a topological sort. Since this is not the case for AND/OR precedence constraints, the algorithms for AND-constraints cannot be applied in that setting.

In the first part of the paper we provide efficient algorithms and structural insights for the more general and complex model of AND/OR precedence constraints. We show that feasibility as well as questions related to generalized transitivity can be solved by applying essentially the same linear-time algorithm. Moreover, we discuss a natural generalization of AND/OR precedence constraints and prove that the same problems become NP-complete in this setting.

The second part of this paper is concerned with the computation of earliest start times if AND/OR precedence constraints are imposed among jobs. We consider different ranges of minimal *time distances* (or time lags) d_{ij} between the start times of two jobs i and j that are coupled within a precedence constraint. For AND/OR precedence constraints, the problem then reduces to finding a solution to a system of *min-max-inequalities* which also has applications in many other fields. We discuss polynomial equivalence to finding optimal strategies for a class of two-person games played on directed graphs. For the case that such time lags are strictly positive (nonnegative) we devise polynomial-time algorithms.

Outline. The paper is organized as follows. After stating some basic requirements in section 2, we discuss feasibility and aspects of transitivity as well as an application thereof in sections 3 to 5. A generalization of AND/OR precedence constraints is considered in section 6. While we consider only the combinatorial structure of AND/OR precedence constraints in sections 3 to 6, we additionally deal with temporal data such as job processing times or time lags between jobs in section 7, where we provide algorithms for computing earliest job start times.

2. Preliminaries. In order to illustrate the presentation we use the following example throughout the paper.

Example 1. Let $V := \{j_1, \dots, j_7\}$ be the set of jobs and $\mathcal{W} := \{w_1 = (\{j_1, j_5\}, j_4)$, $w_2 = (\{j_2, j_6\}, j_4)$, $w_3 = (\{j_4, j_3\}, j_6)$, $w_4 = (\{j_4\}, j_5)$, $w_5 = (\{j_4, j_5, j_6\}, j_7)\}$ the set of waiting conditions.

Graph representation. We use a natural representation of AND/OR precedence constraints by a directed graph D on the set $\mathcal{V} = V \cup \mathcal{W}$ of nodes. The set A of arcs is constructed in the following way: For every waiting condition $w = (X, j) \in \mathcal{W}$, we introduce arcs (i, w), for each $i \in X$, and one additional arc (w, j). Notice that the size of the resulting digraph D is linear in the input size of the problem. The sets V and $\mathcal{W}$ form a bipartition of D. Similar digraphs are used to represent directed hypergraphs; see, e.g., [10] and [9]. For a node $j \in V \cup \mathcal{W}$, we use $in(j)$ and $out(j)$ to denote the sets $\{i \in V \cup \mathcal{W} : (i, j) \in A\}$ and $\{i \in V \cup \mathcal{W} : (j, i) \in A\}$, respectively. We also sometimes use the notation $in_D(j)$ and $out_D(j)$ to stress the underlying digraph D.

The digraph resulting from Example 1 is depicted in Figure 2.1. For the moment, the numbers associated with the arcs can be ignored; they come into play in section 7 when earliest job start times are computed. As usual, a cycle in D is a sequence $(v_0, v_1, \dots, v_k, v_0)$, $v_\ell \in \mathcal{V}$, where $(v_0, v_1, \dots, v_k)$ is a directed path and there exists an

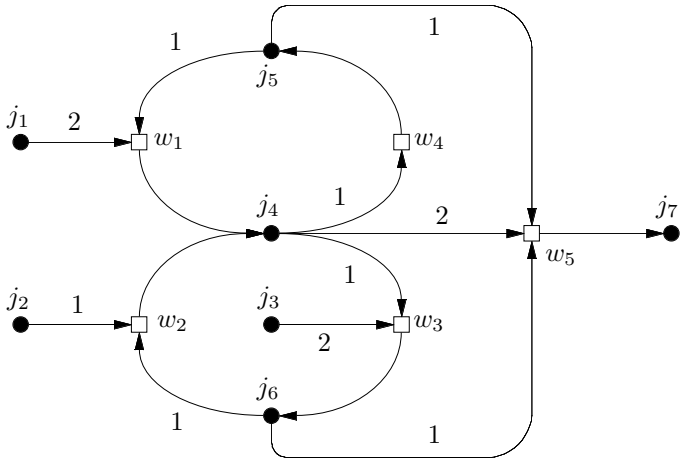


FIG. 2.1. *The digraph resulting from Example* 1. *Circular nodes correspond to jobs (*AND*-nodes), while square nodes represent waiting conditions (*OR*-nodes). Numbers associated with arcs define time lags used in section* 7 *(the time lags of arcs without a number are* 0*).*

arc from v_k to v_0. We also consider *generalized cycles*, which are induced subgraphs D' of D that consist of node sets $V' \subseteq V$ and $\mathcal{W}' \subseteq \mathcal{W}$ such that $in_D(j) \cap \mathcal{W}' \neq \emptyset$ for each $j \in V'$ and $\emptyset \neq in_D(w) \subseteq V'$ for each $w \in \mathcal{W}'$.

Realizations. Given a set V of jobs and a set $\mathcal{W}$ of waiting conditions, an implementation of the corresponding project requires a decision for each waiting condition (X, j): One has to determine a job $i \in X$ that job j should wait for. The entirety of these decisions must lead to a partial order $R = (V, \prec_R)$ on the set V of jobs (the introduction of a cycle would lead to infeasibility) such that

$$\text{for each } (X, j) \in \mathcal{W}, \text{ there exists an } i \in X \text{ with } i \prec_R j. \tag{2.1}$$

Conversely, every partial order R with property (2.1) defines an implementation of the project and is therefore called a *realization* for the set $\mathcal{W}$ of waiting conditions. In what follows, a set $\mathcal{W}$ of waiting conditions is called *feasible* if and only if there exists a realization for $\mathcal{W}$. Since any extension $R' = (V, \prec_{R'})$ of a realization R (i.e., if $i \prec_R j$, then $i \prec_{R'} j$) fulfills property (2.1), R' is also a realization. In particular, a set of AND/OR precedence constraints is feasible if and only if there exists a total order of the jobs which is a realization; we call such a realization *linear*.

Possible linear realizations of Example 1 are, for instance, $j_1 \prec \cdots \prec j_7$ and $j_3 \prec j_6 \prec j_7 \prec j_2 \prec j_1 \prec j_4 \prec j_5$.

3. Feasibility. In order to check whether a given set $\mathcal{W}$ of AND/OR precedence constraints is feasible, we try to construct a linear realization L in a greedy way: While there exists a job $i \in V$ that is not a waiting job of any waiting condition in $\mathcal{W}$, it is inserted at the end of L. Whenever a waiting condition (X, j) becomes satisfied (which is the case if some $i \in X$ is being added to L), (X, j) is deleted from $\mathcal{W}$. Computational details are provided in Algorithm 1. We use a data structure Q to temporarily store jobs from V. Implementing Q as a stack or a queue leads to a linear-time algorithm.

THEOREM 3.1. *A set of* AND/OR *precedence constraints is feasible if and only if the list L obtained from Algorithm* 1 *contains all jobs of V.*

Algorithm 1: Feasibility check of a set of waiting conditions.

Input : A set V of jobs and waiting conditions $\mathcal{W}$.
Output: A list L of jobs from V.

```
Q := ∅; L := ∅;
for jobs j ∈ V do
    a(j) := |{(X, j) ∈ W}|;
    if a(j) = 0 then add j to Q;
while Q ≠ ∅ do
    remove a job i from Q;
    insert i at the end of L;
1   for waiting conditions (X, j) ∈ W with i ∈ X do
        decrease a(j) by 1;
        if a(j) = 0 then add j to Q;
        remove (X, j) from W;
return L;
```

Proof. If L contains all jobs, it follows from the construction of Algorithm 1 that, for each waiting condition $(X, j) \in \mathcal{W}$, there is at least one job $i \in X$ with $i \prec_L j$; therefore, according to (2.1), L is a linear realization. Suppose now that the algorithm returns an incomplete list L although the set of waiting conditions is feasible. Consider a linear realization R and let $j \in V \setminus L$ be minimal with respect to the total order $\prec_R$. Since the algorithm was not able to add j to L, there is a waiting condition $(X, j) \in \mathcal{W}$ with $X \subseteq V \setminus L$. Since R is a realization, there exists a job $i \in X$ with $i \prec_R j$ which is a contradiction of the minimal choice of j. □

As a consequence of Theorem 3.1 we can formulate the following structural characterization of feasible waiting conditions. The lemma appears implicitly already in the work of Igelmund and Radermacher [14] within the context of stochastic resource-constrained project scheduling.

LEMMA 3.2. *A set of* AND/OR *precedence constraints is feasible if and only if there exists no generalized cycle in the associated digraph D.*

Note that Example 1 is feasible (recall that we already stated two linear realizations). However, if $w_1 = (\{j_1, j_5\}, j_4)$ is replaced by $(\{j_5\}, j_4)$, the instance becomes infeasible because $V' = \{j_4, j_5\}$ and $\mathcal{W}' = \{(\{j_5\}, j_4), (\{j_4\}, j_5)\}$ form a generalized cycle.

The following corollary states an algorithmic consequence of the structural insight of Lemma 3.2.

COROLLARY 3.3. *Job $j \in V$ is not contained in the list L returned by Algorithm* 1 *if and only if j is contained in a set $V' \subseteq V$ such that for all $i \in V'$ there is a waiting condition $(X, i) \in \mathcal{W}$ with $X \subseteq V'$.*

In particular, L as a *set* does not depend on the individual jobs chosen from Q in the while-loop of Algorithm 1.

In the proof of Theorem 3.1 we have shown that, for a feasible set of AND/OR precedence constraints $\mathcal{W}$, the list L returned by Algorithm 1 is a linear realization of $\mathcal{W}$. In fact, it is an easy observation that Algorithm 1 can generate every linear realization of $\mathcal{W}$ through an appropriate choice of jobs from Q in the while-loop.

Remark. The problem of checking feasibility of $\mathcal{W}$ can alternatively be solved by transforming it into a satisfiability problem (SAT) where each clause is of *Horn*

type. Such SAT instances are well known to be solvable in linear time; see [7]. The transformation and more details can be found in [26].

4. Detecting implicit AND/OR precedence constraints.

4.1. Problem definition and related work. We now focus on detecting "new" waiting conditions that can be deduced from the given set $\mathcal{W}$ of constraints. For $U \subset V$ and $j \in V \setminus U$, we say that the waiting condition (U, j) is *implied* by $\mathcal{W}$ if and only if

(4.1) for every realization $R = (V, \prec_R)$ of $\mathcal{W}$, there exists some $i \in U$ with $i \prec_R j$.

By property (2.1), this is equivalent to the requirement that adding the waiting condition (U, j) to $\mathcal{W}$ does not change the set of realizations for $\mathcal{W}$. Notice that it is sufficient to claim property (4.1) for every *linear* realization of $\mathcal{W}$.

For traditional precedence constraints the detection of implied waiting conditions is an easy task because the transitive closure which represents all such implicit constraints can be efficiently computed by standard graph algorithms. However, in general, the total number of implicit AND/OR precedence constraints is exponential in the input size of V and $\mathcal{W}$. In particular, it is not possible to compute all implicit constraints efficiently. For the restricted case of AND/OR precedence constraints where the associated digraph D is acyclic, Gillies [11] proposes an algorithm to determine jobs that have to wait for a single job i.

In the context of directed hypergraphs, Ausiello, d'Atri, and Saccà [3] (see also [4]) consider problems similar to those discussed in this section and in section 5 below. However, the results we present are not contained in their work because their definition of implicit hyperarcs differs from our definition of implicit waiting conditions. Their definition is based on three rules which are known as *Armstrong's axioms* within the context of functional dependencies in relational databases (see, e.g., [27]). In particular, [3, Definition 4] does not cover implications that can be deduced from the requirement of feasibility. For instance, in Example 1, the waiting condition $(\{j_1\}, j_4)$ is implied by $\mathcal{W}$, but it is not implied according to [3, Definition 4].

4.2. Result. For a given set $U \subseteq V$ we show that Algorithm 1 can be used to detect all implicit waiting conditions of the form (U, j). For an arbitrary subset $Y \subseteq V$ the set $\mathcal{W}_Y$ of *induced* waiting conditions is given by $\mathcal{W}_Y := \{(X \cap Y, j) \mid (X, j) \in \mathcal{W}, j \in Y\}$. For $(X, j) \in \mathcal{W}$ with $j \in Y$ and $X \cap Y = \emptyset$, the resulting waiting condition $(\emptyset, j) \in \mathcal{W}_Y$ means that job j cannot be planned at all with respect to $\mathcal{W}_Y$; in particular, $\mathcal{W}_Y$ is infeasible in this case.

THEOREM 4.1. *For given $U \subset V$ let L be the output of Algorithm 1 with input $V \setminus U$ and $\mathcal{W}_{V \setminus U}$. The set of waiting conditions of the form (U, j) which are implied by $\mathcal{W}$ is precisely $\{(U, j) \mid j \in V \setminus (L \cup U)\}$.*

The proof is deferred to section 4.3 below. For Example 1 and $U := \{j_2, j_3\}$, the algorithm computes $L = \{j_1\}$, while for $U := \{j_1, j_2\}$ we obtain $L = \{j_3, j_6, j_7\}$. Thus, the waiting condition $(\{j_2, j_3\}, j_7)$ is implied by $\mathcal{W}$, while $(\{j_1, j_2\}, j_7)$ is not.

We can directly deduce the following corollary.

COROLLARY 4.2. *Given $U \subset V$, the set of waiting conditions of the form (U, j) that are implied by $\mathcal{W}$ can be computed in linear time.*

4.3. Correctness. We next state some rather technical lemmas which directly show the validity of Theorem 4.1. The theorem can alternatively be proved by a simpler argumentation (similar to the proof of Theorem 3.1), but we need the lemmas

to establish other results in section 5 below. In addition, with the extended argumentation, we are able to strengthen Theorem 4.1 slightly (see Corollary 4.6). The following definition will be useful throughout the discussion: For a given feasible set $\mathcal{W}$ of waiting conditions and a set $U \subseteq V$ let

$$\begin{aligned} Y_U &:= \{j \in V \setminus U \mid (U,j) \text{ is not implied by } \mathcal{W}\}, \\ Z_U &:= \{j \in V \setminus U \mid (U,j) \text{ is implied by } \mathcal{W}\} = V \setminus (U \cup Y_U) \ . \end{aligned}$$

LEMMA 4.3. *Let $\mathcal{W}$ be a feasible set of waiting conditions and let $U \subseteq V$. Then there exists a (linear) realization $R = (V, \prec_R)$ of $\mathcal{W}$ such that Y_U is an order ideal of R; i.e., R "starts" with the jobs in Y_U.*

Proof. Let $R' = (V, \prec_{R'})$ be a linear realization of $\mathcal{W}$ that maximizes the cardinality of the largest order ideal J of R' with $J \subseteq Y_U$. To show that $J = Y_U$, by contradiction, we assume that there is a job $j' \in Y_U \setminus J$. Since, by definition of Y_U, the waiting condition (U, j') is not implied by $\mathcal{W}$, there is a linear realization $R = (V, \prec_R)$ of $\mathcal{W}$ with $j' \prec_R U$ (j' precedes all elements in U). Let $j \in Y_U \setminus J$ be minimal with respect to R. By maximality of J, job j cannot be moved to the position directly after J in R' without violating a waiting condition. Thus, there exists $(X, j) \in \mathcal{W}$ with $X \subseteq V \setminus J$. By (2.1), there exists an $i \in X$ with $i \prec_R j$. Notice that we have $i \notin U$ because $i \prec_R j \preceq_R j' \prec_R U$. Moreover, due to the minimal choice of j and the fact that $i \notin J$ it follows that $i \notin Y_U$. As a consequence we obtain $i \in X \setminus (U \cup Y_U)$. By definition of Z_U, this yields $i \in Z_U$. Thus, the waiting condition (U, i) is implied by $\mathcal{W}$, which is a contradiction of $i \prec_R j \preceq_R j' \prec_R U$. □

Let us call a set $Y \subseteq V$ *feasible with respect to* $\mathcal{W}$ if and only if the induced set $\mathcal{W}_Y$ of waiting conditions is feasible. The result in Corollary 3.3 can then be restated as follows.

COROLLARY 4.4. *Algorithm* 1 *returns the unique maximal feasible subset of V with respect to $\mathcal{W}$.*

In conjunction with the following lemma, Corollary 4.4 provides an efficient way of detecting waiting conditions implied by $\mathcal{W}$. This concludes the proof of Theorem 4.1 (in fact, the lemma essentially is a reformulation of Theorem 4.1).

LEMMA 4.5. *Let $\mathcal{W}$ be a feasible set of* AND/OR *precedence constraints, $U \subset V$, and $j \in V \setminus U$. Then the waiting condition (U, j) is implied by $\mathcal{W}$ if and only if j is not contained in the unique maximal feasible subset of $V \setminus U$ with respect to $\mathcal{W}_{V \setminus U}$.*

Proof. We have to show that Y_U is the unique maximal feasible subset F of $V \setminus U$ with respect to $\mathcal{W}_{V \setminus U}$. By Lemma 4.3, there exists a linear realization R of $\mathcal{W}$ starting with the jobs in Y_U. This induces a linear realization of $\mathcal{W}_{Y_U}$ and consequently, by definition, Y_U is feasible with respect to $\mathcal{W}$. Moreover, since $\mathcal{W}_{Y_U} = (\mathcal{W}_{V \setminus U})_{Y_U}$, the subset Y_U is feasible with respect to $\mathcal{W}_{V \setminus U}$, which yields $Y_U \subseteq F$.

To show that $F \subseteq Y_U$, by contradiction, assume that $F \setminus Y_U \neq \emptyset$. Since $F \cap U = \emptyset$ we then have $F \cap Z_U \neq \emptyset$. Let $R' = (F, \prec_{R'})$ be a linear realization of $\mathcal{W}_F$ and choose $i \in F \cap Z_U$ minimal with respect to R'. Since $i \in Z_U$, by definition of Z_U, the waiting condition (U, i) is implied by $\mathcal{W}$. Therefore, moving job i to the position directly after Y_U in R violates a waiting condition $(X, i) \in \mathcal{W}$ with $X \subseteq U \cup Z_U$. Let us consider the induced waiting condition $(X \cap F, i) \in \mathcal{W}_F$. It follows from the minimal choice of i that $i' \notin F \cap Z_U$ for all i' with $i' \prec_{R'} i$. With $F \cap U = \emptyset$, this implies $i' \notin X \cap F$, which is a contradiction of the fact that R' is a realization for $\mathcal{W}_F$. □

Finally, notice that there may exist (implicit) waiting conditions (X, j) inside the considered set U, i.e., $X \subset U$ and $j \in U \setminus X$. Theorem 4.1 can be strengthened in the following way. Consider the situation after the execution of Algorithm 1 with input

$V \setminus U$ and $\mathcal{W}_{V\setminus U}$ and let L be the resulting list of jobs. Furthermore, let $U' \subseteq U$ denote the set of jobs from U that can be added to L without violating any waiting condition of $\mathcal{W}$.

COROLLARY 4.6. *For given $U \subseteq V$ the set of waiting conditions (U', j) which is implied by $\mathcal{W}$ is precisely $\{(U', j) \mid j \in V \setminus (L \cup U')\}$.*

Proof. We show that the maximal feasible subsets F and F' of $V \setminus U$ and $V \setminus U'$, respectively, coincide. The corollary then follows from Lemma 4.5. It is clear that $F \subseteq F'$ since $U' \subseteq U$. Conversely, suppose by contradiction that $F' \setminus F \neq \emptyset$. Denote by $R = (V, \prec_R)$ and $R' = (V, \prec_{R'})$ linear realizations of $\mathcal{W}$ where F and F' are order ideals, respectively (the existence of R and R' follows from Lemmas 4.3 and 4.5). Let $j \in F' \setminus F$ be the smallest job in $F' \setminus F$ with respect to R'. Since $j \notin F$, moving job j in R to the position directly after F violates a waiting condition (X, j) with $X \cap F = \emptyset$. Since R' is a realization there must exist some $i \in X$ with $i \prec_{R'} j$. But $i \in F' \setminus F$, which contradicts the minimal choice of j. ∎

The results presented in this section turn out to be useful in the context of minimal representations of AND/OR precedence constraints discussed in section 5. Besides this, Corollary 4.6 led to a considerable speedup of computation time within branch-and-bound procedures for stochastic resource-constrained project scheduling; see [26] for details.

5. Minimal representations of AND/OR precedence constraints. While for traditional precedence constraints a minimal representation without redundancies is given by the transitive reduction and can be computed by simply removing redundant (i.e., transitive) constraints, the situation is slightly more complicated for AND/OR precedence constraints. In order to obtain a *unique* minimal representation, it is not sufficient to iteratively remove redundant waiting conditions that are implied by the others.

DEFINITION 5.1. *A set $\mathcal{W}$ of waiting conditions is called* minimal *if*

(i) *no waiting condition $(X, j) \in \mathcal{W}$ is implied by $\mathcal{W} \setminus \{(X, j)\}$, and*

(ii) *for each waiting condition $(X, j) \in \mathcal{W}$, the set X is minimal with respect to inclusion; i.e., for all $i \in X$, the waiting condition $(X \setminus \{i\}, j)$ is not implied by $\mathcal{W}$.*

Two sets $\mathcal{W}$ and $\mathcal{W}'$ of waiting conditions are called equivalent *if their sets of (linear) realizations coincide. Moreover, if $\mathcal{W}'$ is minimal, then $\mathcal{W}'$ is called a* minimal reduction *of $\mathcal{W}$.*

The set $\mathcal{W}$ from Example 1 is not minimal: If the waiting condition $(\{j_4, j_5, j_6\}, j_7)$ is replaced by $(\{j_4, j_6\}, j_7)$, the resulting instance is equivalent to Example 1. This follows from waiting condition $(\{j_4\}, j_5)$, which ensures that whenever $j_5 \prec_R j_7$ in some realization $R = (V, \prec_R)$ we also have $j_4 \prec_R j_5$ and $j_4 \prec_R j_7$. Note that if we additionally replace $(\{j_1, j_5\}, j_4)$ by $(\{j_1\}, j_4)$, the resulting set of waiting conditions is minimal (and still equivalent to Example 1).

THEOREM 5.2. *Each feasible set of waiting conditions has a unique minimal reduction.*

To prove the theorem we need the following technical lemma.

LEMMA 5.3. *Let $\mathcal{W}$ be a feasible set of waiting conditions with $(U, j) \in \mathcal{W}$. The waiting condition (U, j) is implied by $\mathcal{W}' := \mathcal{W} \setminus \{(U, j)\}$ if and only if there exists some $(X, j) \in \mathcal{W}'$ with $X \subseteq U \cup Z_U$.*

Proof. If (U, j) is implied by $\mathcal{W}'$, then any ordering of V where Y_U is an ideal and j is placed directly after Y_U is not a realization of $\mathcal{W}'$. Thus, there exists some $(X, j) \in \mathcal{W}$ with $X \subseteq U \cup Z_U$. Contrarily, suppose that there exists some $(X, j) \in \mathcal{W}'$

Algorithm 2: Computation of a minimal reduction.

Input : A set V of jobs and waiting conditions $\mathcal{W}$.
Output: A minimal reduction of $\mathcal{W}$

```
for each (U, j) ∈ W do
    L := call Algorithm 1 with input V \ U and W_{V\U} and
         compute a(i) for each i ∈ V;
    if a(j) > 1 then
        delete (U, j) from W;
    else for i ∈ U do
        if a(i) > 0 then delete i from U;
return W;
```

with $X \subseteq U \cup Z_U$ but (U, j) is not implied by $\mathcal{W}'$. Then there exists some $h \in X \setminus U$ and a linear realization $R' = (V, \prec_{R'})$ of $\mathcal{W}'$ with $Y_U \prec_{R'} h \prec_{R'} j \prec_{R'} U$. Since $\mathcal{W}$ is feasible, there exists some $i \in U$ that can be moved to the position directly after h without violating a waiting condition of $\mathcal{W}'$ (this follows from Corollary 4.6). In addition, the resulting linear realization $R = (V, \prec_R)$ satisfies the waiting condition (U, j) and thus should be a realization with respect to $\mathcal{W}$. However, we have $h \prec_R i \prec_R j \prec_R U \setminus \{i\}$, which is a contradiction of $h \in Z_U$. □

Proof of Theorem 5.2. Let $\mathcal{W}$ and $\mathcal{W}'$ be equivalent and both minimal. It suffices to show that $(U, j) \in \mathcal{W}$ implies $(U, j) \in \mathcal{W}'$. By Lemma 4.3, there exists a linear realization R of $\mathcal{W}$ starting with Y_U. Since the order obtained by moving j to the position directly after Y_U in R is not a realization, there exists a waiting condition $(X, j) \in \mathcal{W}'$ with $X \subseteq U \cup Z_U$. We show next that $U \subseteq X$. By minimality of $\mathcal{W}'$ this implies $X = U$, which concludes the proof.

Assume that $X \not\supseteq U$ and let $i \in U \setminus X$. We obtain a linear realization R' by moving i to the position directly after Y_U in R; otherwise, there exists a waiting condition $(Z, i) \in \mathcal{W}$ with $Z \subseteq U \cup Z_U$. Since all jobs in Z_U have to wait for a job in U, the waiting condition $(U \setminus \{i\}, i)$, and thus $(U \setminus \{i\}, j)$, is implied by $\mathcal{W}$, which is a contradiction of the minimality of $\mathcal{W}$.

Since moving j to the position directly after $Y_U \cup \{i\}$ in R' violates the waiting condition (X, j), there exists a waiting condition $(Z, j) \in \mathcal{W}$ with $Z \subseteq (U \setminus \{i\}) \cup Z_U$. However, by Lemma 5.3, the set $\mathcal{W} \setminus \{(U, j)\}$ implies the waiting condition (U, j), which is a contradiction of the minimality of $\mathcal{W}$. □

Let us next consider the following straightforward polynomial-time algorithm to compute a minimal reduction of a set $\mathcal{W}$ of waiting conditions. For each $(X, j) \in \mathcal{W}$, apply Algorithm 1 with input $V \setminus X$ and $\mathcal{W}_{V \setminus X}$. If, besides (X, j), some other waiting condition prevents j from being added to L, then remove (X, j) from $\mathcal{W}$. Otherwise, remove all i from X, which cannot be added to L because some waiting condition of $\mathcal{W}$ is violated. Finally, output the resulting set of waiting conditions. An implementation of this rough scheme is given in Algorithm 2. There, $a(j)$, $j \in V$, denotes the number of waiting conditions of the form (X, j) that are left in $\mathcal{W}_{V \setminus U}$ after Algorithm 1 was called with input $V \setminus U$ and $\mathcal{W}_{V \setminus U}$. Notice that $a(j)$ is computed within the execution of Algorithm 1. In the following theorem we prove the correctness of the algorithm (as defined earlier, A is the set of arcs in the digraph induced by $\mathcal{W}$).

THEOREM 5.4. *Algorithm* 2 *computes the minimal reduction of a set* $\mathcal{W}$ *of waiting conditions in* $O(|\mathcal{W}| \cdot |A|)$ *time.*

Proof. We first show that, through the procedure, the transformed set of waiting conditions is equivalent to $\mathcal{W}$ given as input. We then argue that, once the algorithm has finished, the obtained set of waiting conditions is minimal.

Denote by $\mathcal{W}^k$, $k \in \{1, \dots, |\mathcal{W}|\}$, the set of waiting conditions after the kth iteration of the outer for-loop of Algorithm 2. Furthermore, let $\mathcal{W}^0 := \mathcal{W}$. Suppose that some (U, j) is removed from $\mathcal{W}^{k-1}$ in the kth iteration of the algorithm. Since $a(j) > 1$ in the kth iteration, there exists a waiting condition $(X, j) \in \mathcal{W}^{k-1}$ with $X \neq U$ and $X \subset U \cup Z_U$. With Lemma 5.3, (U, j) is implied by $\mathcal{W}^k = \mathcal{W}^{k-1} \setminus \{(U, j)\}$ and can thus be deleted from $\mathcal{W}^{k-1}$. Now assume that, in (U, j), some job i was deleted from U in the kth iteration. Then $a(i) > 0$ and with Corollary 4.6 it follows that $(U \setminus \{i\}, i)$ is implied by $\mathcal{W}^{k-1}$. Together with (U, j) this shows that $(U \setminus \{i\}, j)$ is implied by $\mathcal{W}^{k-1}$. Thus, $\mathcal{W}^{k-1}$ is equivalent to $\mathcal{W}^k$ for all $k \in \{1, \dots, |\mathcal{W}|\}$, which directly implies that $\mathcal{W}$ and $\mathcal{W}' := \mathcal{W}^{|\mathcal{W}|}$ are equivalent.

We now show that $\mathcal{W}'$ is minimal. Let us first suppose that some $(U, j) \in \mathcal{W}'$ is implied by $\mathcal{W}' \setminus \{(U, j)\}$. Then, by Lemma 5.3, there exists another waiting condition $(X, j) \in \mathcal{W}' \setminus \{(U, j)\}$ with $X \subseteq U \cup Z_U$. Notice that Z_U in dependence of $\mathcal{W}' \setminus \{(U, j)\}$ and all $\mathcal{W}^k$, $k \in \{0, \dots, |\mathcal{W}|\}$, is constant because the associated sets of realizations coincide. The waiting conditions (U, j) and (X, j) have been constructed in some iterations k and k', respectively, in which waiting conditions $(U', j) \in \mathcal{W}$ with $U \subseteq U'$ and $(X', j) \in \mathcal{W}$ with $X \subseteq X'$ have been treated by the algorithm. If $(X, j) \in \mathcal{W}^{k-1}$, then, by Lemma 5.3, (U', j) would have been removed from $\mathcal{W}^{k-1}$. Consequently, $k' > k$. Since U was obtained from U' (in the kth iteration of the algorithm), by Corollary 4.6, there exists a linear realization R which starts with Y_U and is followed first by an *arbitrary* job $i \in U$ and then by job j. Since, by assumption, $\mathcal{W}^{k-1}$ and $\mathcal{W}' \setminus \{(U, j)\}$ are equivalent, (X, j) must be respected by R; hence $U \subseteq X$. But then (X', j) is deleted in iteration $k' > k$, a contradiction. Next, suppose that $\mathcal{W}'$ contains a waiting condition (U, j) such that, for some $i \in U$, the waiting condition $(U \setminus \{i\}, j)$ is implied by $\mathcal{W}'$. Since i was not removed from U' in the kth iteration of the algorithm, it follows from Corollary 4.6 that $(U' \setminus \{i\}, j)$ is not implied by $\mathcal{W}^{k-1}$. Thus there exists a linear realization $R = (V, \prec_R)$ of $\mathcal{W}^{k-1}$ with $j \prec_R (U' \setminus \{i\})$ and in particular $j \prec_R (U \setminus \{i\})$. Since R is also a realization for $\mathcal{W}'$, the waiting condition $(U \setminus \{i\}, j)$ is not implied by $\mathcal{W}'$—a contradiction.

The above argumentation shows that $\mathcal{W}'$ is minimal and thus Algorithm 2 computes a minimal reduction of $\mathcal{W}$. Finally, the running time follows from the fact that Algorithm 1 is called $|\mathcal{W}|$ times. ∎

Notice that the cardinality of a minimal set of waiting conditions might still be exponential in the number of jobs $|V|$: Let $V = \{1, 2, \dots, 2\ell + 1\}$; in order to model the constraint that job $2\ell + 1$ can be planned only after at least ℓ other jobs, we need exactly $\binom{2\ell}{\ell}$ waiting conditions.

6. An NP-complete generalization. Suppose that we generalize the definition of waiting conditions from (X, j), $X \subset V$, $j \in V \setminus X$ to (X, X') with $X, X' \subset V$ and $X \cap X' = \emptyset$. The generalized waiting condition (X, X') is fulfilled if at least one job $j \in X'$ is waiting for at least one job $i \in X$. We show in the theorem below that the problems considered in sections 3 and 4 become NP-complete in this generalized setting.

THEOREM 6.1. *Given a set of jobs with generalized waiting conditions, it is NP-complete to decide whether or not a waiting condition* $(\{i\}, \{j\})$ *is implied for two jobs* i *and* j.

Proof. We construct a reduction from the satisfiability problem SAT. Given an instance of SAT, we introduce for each Boolean variable x two jobs which correspond to the two literals x and $\bar{x}$ (negation of x); to keep notation simple, we denote these jobs also by x and $\bar{x}$. Moreover, for each clause C we introduce a corresponding job (also denoted by C) and a waiting condition $(X_C, \{C\})$, where X_C denotes the set of literals in clause C; in other words, job C may not be started before at least one job corresponding to a literal of clause C has been completed. Finally, we introduce two additional jobs s and t together with the following waiting conditions: For each variable x, at least one of the jobs x and $\bar{x}$ has to wait for s; i.e., we have the waiting condition $(\{s\}, \{x, \bar{x}\})$. For each clause C, job t has to wait for the corresponding job, which is given by the waiting condition $(\{C\}, \{t\})$.

It is easy to check that in the constructed scheduling instance job t has to wait for job s if and only if the underlying instance of SAT does not have a satisfying truth assignment. If there is a satisfying truth assignment, then we can construct a linear realization where t precedes s in the following way: First we take all jobs corresponding to literals with value "true" in an arbitrary order; next we append all jobs corresponding to clauses in some order; afterwards we add t, then s, and finally all remaining jobs corresponding to literals with the value "false." On the other hand, if there is a linear realization where t precedes s, we can define a corresponding satisfying truth assignment in the following way: For each variable x, assign x the value true (false) if the job corresponding to x ($\bar{x}$) precedes s; notice that at most one of the two cases can happen: if neither job x nor $\bar{x}$ precedes s, we assign an arbitrary value to the variable x. □

As a consequence of Theorem 6.1, we obtain that the problem of deciding feasibility for a set of generalized waiting conditions is also NP-complete. To see this, add to the construction in the proof the waiting condition $(\{t\}, \{s\})$. Then the given instance of SAT is feasible if and only if the constructed instance of the scheduling problem with generalized waiting conditions is feasible.

7. Computing earliest job start times. This section is concerned with the computation of earliest job start times subject to AND/OR precedence constraints. The underlying problem is to find a solution to a system of *min-max-inequalities*. There are several other applications of such systems of inequalities; we will mention some of them below.

7.1. Problem definition and related work. For the remainder of the paper we assume that together with each waiting condition $w = (X, j) \in \mathcal{W}$ and each job $i \in X$ we are given an integral time lag $-M < d_{iw} < M$, $M \geqslant 0$. We aim at finding a vector of earliest start times $S = (S_1, \ldots, S_n)$ such that for each waiting condition $(X, j) \in \mathcal{W}$ the constraint

$$S_j \;\geqslant\; \min_{i \in X}(S_i + d_{iw}) \tag{7.1}$$

is satisfied. Job processing times p_i can be modeled by setting $d_{iw} := p_i$ for all $w = (X, j)$ with $i \in X$. Negative values d_{iw} represent so-called *maximal time lags* that define latest possible start times of jobs $i \in X$ relative to j.

In order to simplify the presentation, we sometimes interpret nodes of the digraph D that represent waiting conditions as dummy jobs. We then assume that the vector S also contains start times of these dummy jobs and constraint (7.1) is replaced by

$$S_w \;\geqslant\; \min_{i \in X}(S_i + d_{iw}) \qquad \text{and} \qquad S_j \;\geqslant\; S_w.$$

We call the jobs in V AND-*nodes* and the jobs in $\mathcal{W}$ OR-*nodes* of the digraph D. We assume that a dummy AND-node s precedes all other AND-nodes; i.e., we introduce a waiting condition $(\{s\}, j)$ for all $j \in V$. In D, time lags can easily be integrated by associating each d_{iw} as a weight to the arc (i, w); see Figure 2.1.

The problem of finding earliest start times can then be formulated on D as follows: Find a componentwise minimal schedule $S \in \mathbb{Z}^{|\mathcal{V}|}$ fulfilling $S_s \geqslant 0$ and

$$\text{(ES)} \qquad \begin{aligned} S_j &\geqslant \max_{(w,j)\in A} (S_w + d_{wj}), \qquad j \in V, \\ S_w &\geqslant \min_{(j,w)\in A} (S_j + d_{jw}), \qquad w \in \mathcal{W}. \end{aligned}$$

In the above formula we added the term d_{wj} for symmetry reasons. Without loss of generality we assume that $d_{wj} = 0$. The case $d_{wj} \neq 0$ can be handled by replacing $d_{wj} \neq 0$ by 0 and d_{iw} by $d_{iw} + d_{wj}$ for all $i \in in(w)$. Besides schedules $S \in \mathbb{Z}^{|\mathcal{V}|}$ we also consider *partial schedules* $S \in (\mathbb{Z} \cup \{\infty\})^{|\mathcal{V}|}$ where the start time of a job may be infinite, meaning that the job is not planned. As usual, a (partial) schedule fulfilling the constraints of (ES) is called *feasible*. In particular, the partial schedule $S = (\infty, \ldots, \infty)$ fulfills all inequalities of (ES) and is thus feasible. Moreover, it is easy to see that if S' and S'' are feasible partial schedules, then their componentwise minimum $S := \min\{S', S''\}$ is also feasible. In particular, there always exists a (unique) componentwise minimal partial schedule S^*, called the *optimal* partial schedule (notice that $S^* \geqslant 0$ for all AND-nodes). It follows that, instead of considering the above system of inequalities, we alternatively may consider the corresponding system of equations (which is obtained from (ES) by replacing each "$\geqslant$" by "$=$").

Presuming different restrictions on the range of arc weights, several algorithms have been suggested to solve (ES). Note that all restrictions on arc weights are meant to refer to arcs (j, w) between AND-nodes j and OR-nodes w only. For the case of nonnegative arc weights without cycles of zero length in D, a modification of Dijkstra's shortest path algorithm can be applied. An algorithm suggested by Knuth [16] has running time $O(|\mathcal{V}| \log |\mathcal{V}| + |A|)$. Other approaches are proposed in [6], [10], and, in the context of resource-constrained project scheduling, [14] (see also [22]). Levner, Sung, and Vlach [18] consider a generalized model of AND/OR precedence constraints where a so-called *threshold value* $1 \leqslant \ell_w \leqslant |X|$ is associated with each waiting condition $w = (X, j)$, indicating that j may start if at least ℓ_w jobs from X have been completed. They show that Dijkstra's shortest path algorithm can also be generalized to solve their model (with positive arc weights). For a discussion of the case with nonnegative arc weights *and* cycles of zero length we refer to subsection 7.4. The general case $-M < d_{jw} < M$ is a frequently studied problem with applications in many different areas, e.g., *game theory* [29] and *interface timing verification* (see [24] and [20]). Moreover, there are applications stemming from online optimization; see [29, section 7] for a collection of examples. Interestingly, although a pseudopolynomial algorithm to solve this case of (ES) is easily obtained, no algorithm polynomial in $|\mathcal{V}|$ and $\log(M)$ is currently known.

7.2. Arbitrary arc weights. In this section we study the case of arbitrary arc weights $-M < d_{jw} < M$.

7.2.1. Feasibility. For arbitrary arc weights the feasibility results stated in section 3 are no longer valid. They are based on the requirement that all $d_{jw} = p_j > 0$, and, consequently, an AND-node j can start if and only if for all $(X, j) \in \mathcal{W}$ at least one $i \in X$ has previously been started (compare with condition (2.1)). However, if we

allow $d_{jw} \leqslant 0$, this is no longer the case. In what follows we derive a necessary and sufficient feasibility criterion for (ES) with arbitrary arc weights which generalizes the feasibility criterion given in Lemma 3.2. For the remainder of the section we call a set $\mathcal{W}$ of waiting conditions *feasible* if and only if there exists a feasible schedule for (ES).

Before we derive the criterion, we discuss how a given instance can be simplified without changing the optimal partial schedule S^*. First, we make the problem more restrictive by removing all but one incoming arc of each OR-node w. If the remaining arc (j, w) fulfills $S_j^* + d_{jw} \leqslant S_w^*$, clearly, all inequalities of (ES) are still satisfied. Consequently, S^* and the optimal (partial) schedule of the more restrictive instance coincide. In a similar fashion we can remove all but one incoming arc (w, j) of each AND-node j without changing the earliest start times. However, removing such arcs means relaxing the problem, and some more work has to be done in order to obtain the desired result.

LEMMA 7.1. *For each digraph D representing a set of* AND/OR *precedence constraints, there exists a subdigraph $\bar{D}$ on the same set $\mathcal{V}$ of nodes with $|in_{\bar{D}}(j)| \leqslant 1$ for all* AND*-nodes $j \in V$ such that $S^* = \bar{S}^*$, where $\bar{S}^*$ denotes the optimal (partial) schedule of $\bar{D}$.*

Proof. We construct $\bar{D}$ by iteratively removing arcs (w, j) from D that do not affect the earliest start time of the AND-node j. By contradiction, assume that once all such arcs have been removed, there is some AND-node j with $|in_{\bar{D}}(j)| > 1$. Thus, removing any incoming arc of j reduces the earliest start time of j. Denote by S^1 and S^2 the optimal (partial) schedules obtained if two different incoming arcs (w_1, j) and (w_2, j) are removed from $\bar{D}$; then $S_j^* > S_j^1$ and $S_j^* > S_j^2$. Without loss of generality let $S_j^1 \leqslant S_j^2$; we define a new (partial) schedule S through

$$S_i \;:=\; \min\{S_i^1 + S_j^2 - S_j^1, S_i^2\} \qquad \text{for all } i \in \mathcal{V}.$$

By definition, $S \leqslant S^2$ and $S_j = S_j^2$. For each arc (w, j) with $w \neq w_2$ we have $S_j^2 \geqslant S_w^2$, which yields $S_j = S_j^2 \geqslant S_w^2 \geqslant S_w$. For $w = w_2$ we get $S_j = S_j^1 + S_j^2 - S_j^1 \geqslant S_w^1 + S_j^2 - S_j^1 \geqslant S_w$. We obtain $S_j \geqslant \max_{(w,j)\in A}(S_w)$. Furthermore, S also fulfills all other inequalities of (ES), because both $S^1 + S_j^2 - S_j^1$ and S^2 fulfill the inequalities and so does its minimum S. Consequently, S is a feasible (partial) schedule, which is a contradiction of the minimality of S^* since $S_j < S_j^*$. ∎

For the subsequent presentation, recall the definition of a (generalized) cycle from section 2. Note that we assume all cycles to be directed cycles.

COROLLARY 7.2. *Let $\bar{D}$ be as in Lemma* 7.1*. Then all cycles in $\bar{D}$ have strictly positive length.*

Proof. Assume that there is a cycle $(w_1, j_1, w_2, j_2, \dots, w_k, j_k, w_1)$ of nonpositive length in $\bar{D}$. By definition of $\bar{D}$, (w_ℓ, j_ℓ) is the only incoming arc for node j_ℓ, $\ell = 1, \dots, k$. Thus, one can construct a feasible partial schedule for $\bar{D}$ satisfying

$$S_{w_1} = S_{j_1} = -1 \qquad \text{and} \qquad S_{w_\ell} = S_{j_\ell} = -1 + \sum_{q=2}^{\ell} d_{j_{q-1}w_q} \quad \text{for } \ell = 2, \dots, k.$$

With Lemma 7.1, this is also possible for the original digraph D, which yields a contradiction of the requirement $S_{j_1}^* \geqslant 0$. ∎

LEMMA 7.3. *A set of* AND/OR *precedence constraints with arbitrary arc weights is feasible if and only if each generalized cycle in D contains a cycle of nonpositive length.*

Proof. Let C be a generalized cycle which contains only cycles of positive length and suppose that some node $v \in C$ can be scheduled at $S_v < \infty$. Since v has at least one incoming arc, there must exist a node $u \in in(v)$ with $S_v \geqslant S_u + d_{uv}$. Iterating this argument, since $|C|$ is finite, we obtain a cycle in C with nonpositive length—a contradiction.

Conversely, suppose that the given instance is infeasible. Let $Z \neq \emptyset$ denote the set of nodes whose earliest start times are ∞. By Lemma 7.1, we can relax the problem by removing all but one incoming arc of each AND-node such that the earliest start times remain unchanged for the resulting digraph $\bar{D}$. Remove all OR-nodes from Z whose out-degree is 0 in $\bar{D}$ and denote the resulting set of nodes by Z'. Then Z' induces a generalized cycle C in D. Moreover, by definition of Z', every cycle in C is also contained in $\bar{D}$ and therefore has positive length by Corollary 7.2. ∎

Lemma 7.3 reduces to Lemma 3.2 if all the arc weights d_{jw} are strictly positive. Lemma 7.3 enables us to show that the decision problem of (ES) is in both NP and co-NP. The decision problem corresponding to (ES) is to decide whether or not a feasible schedule $S < \infty$ for (ES) exists.

LEMMA 7.4. *The decision problem corresponding to* (ES) *is in NP ∩ co-NP.*

Proof. It is clear that the decision problem corresponding to (ES) is in NP because, for a given feasible schedule S of some instance I, it is easy to verify all constraints of $\mathcal{W}$. Moreover, it follows from Lemma 7.3 that the decision problem corresponding to (ES) is in co-NP. We can guess a generalized cycle violating the condition in Lemma 7.3, which can be verified in polynomial time by, for example, some standard minimum mean weight cycle algorithm. ∎

7.2.2. A simple pseudopolynomial time algorithm. For the case of (ES) with arbitrary arc weights several pseudopolynomial algorithms (partly independent of each other) have been proposed; see, e.g., [5] and [25] as well as [24] and [29]. A very simple (pseudopolynomial) algorithm is as follows: First, initialize $S_j := 0$ for all $j \in V$. Then, while S violates some waiting condition $w = (X, j) \in \mathcal{W}$, set $S_j := \min_{i \in X}(S_i + d_{iw})$. If S_j becomes larger than a given time horizon T, then stop and return that the given instance is infeasible. The time horizon T can be chosen as $T := \sum_{j \in V}(\max_{w \in out(j)} |d_{jw}|)$. One can show straightforwardly by induction that $S \leqslant S^*$ in each iteration of the algorithm. If the recurrence stops with $S_j \leqslant T$ for all $j \in V$, then all constraints are obviously fulfilled. Hence $S \geqslant S^*$ and thus we have $S = S^*$. Moreover, at least one start time of a job is increased by 1 in each iteration. Thus the number of iterations is $O(|V| \cdot T)$. Finding a violated waiting condition obviously requires at most $O(|A|)$ time and thus the total complexity is $O(|V| \cdot |A| \cdot T)$. Note that for the special case that D is acyclic, earliest job start times can easily be computed in linear time along a topological sort. Moreover, if each AND-node (OR-node) has at most one incoming arc, node start times can be computed by, for example, a slight modification of the Bellman–Ford shortest (longest) path algorithm in time $O(|V| \cdot |A|)$.

7.2.3. A game-theoretic application. We next consider a class of two-player games played on bipartite directed graphs which are directly related to the problem (ES). There exists substantial literature on different variations of this game; see, e.g., [29], [8], [15], [28], and references therein. Each player is identified with one of the node partitions of the graph. The game starts at a fixed node j_0 and the player associated with that node chooses an incoming arc (w_0, j_0). Then, at node w_0, the other player chooses an incoming arc (j_1, w_0) and so on. The objective and the stopping criterion depend on the considered variation of the game.

One variant is the so-called *mean payoff game* (MPG), where an integer weight is associated to each arc of the digraph. Furthermore, it is assumed that each node has at least one incoming arc. The MPG is finished as soon as the path P resulting from the game contains a cycle and the *outcome* ν of the game is the mean weight of the arcs of that cycle. One player wants to maximize the outcome, while the other player wants to minimize it. It has been shown by Ehrenfeucht and Mycielski [8] that both players have positional optimal strategies; that is, the decisions of both players depend on neither previous choices nor the start node j_0. In the following we always assume that j_0 is associated with the maximization player.

The decision problem corresponding to MPG is to decide whether the outcome of the game is positive. Zwick and Paterson [29] have noted that this problem is in NP $\cap$ co-NP. Even more, Jurdziński [15] showed that the problem is in UP $\cap$ co-UP. It seems to be intuitively clear that MPG and (ES) are closely related. We next show that this is indeed the case.

LEMMA 7.5. *The decision problems corresponding to MPG and* (ES) *are polynomially equivalent.*

Proof. Given an instance of (ES), we construct an instance of MPG in the following way. First, we add an additional job t and a waiting condition $w_j = (\{j\}, t)$ with $d_{jw_j} = 0$ for every job $j \in V$; moreover, we add a waiting condition $w = (\{t\}, s)$ with $d_{tw} = -T$, where T is the time horizon discussed in section 7.2.2. Notice that there exists a feasible schedule for the original instance of (ES) if and only if the earliest start time of the new job t is finite. The game digraph D is now the digraph representing the new scheduling instance. The starting node is $j_0 := t$ and the maximization player starts. We show that the set of AND/OR precedence constraints is feasible if and only if $\nu \leqslant 0$.

Only if: Based on an optimal schedule $S^* < \infty$, we give a strategy for the minimization player which ensures $\nu \leqslant 0$: In each OR-node w, choose an incoming arc (j, w) with $S^*_j + d_{jw} = S^*_w$. Then, for two vertices v_1 and v_2 on the path formed by the game, the weight of the (directed) subpath from v_1 to v_2 is at most $S^*_{v_2} - S^*_{v_1}$ (for each arc (w, j) on the path we have $S^*_j \geqslant S^*_w$ and for each arc (j, w) on the path we have $S^*_w = S^*_j + d_{jw}$). In particular, the length of the cycle terminating the game is at most 0 (choose $v_1 = v_2$).

If: For an infeasible scheduling instance it follows from Lemma 7.3 that there exists a generalized cycle C in D which contains only cycles of positive length. Without loss of generality, C contains the node t (if t is not in C, then consider the generalized cycle where all waiting conditions $(\{j\}, t)$ with j in C are added to C). We give a strategy for the maximization player which ensures $\nu > 0$: In each step, choose an arc which starts at a node in C. Such an arc always exists by the definition of generalized cycles. Moreover, again by the definition of generalized cycles, the minimization player is not able to leave C. This yields $\nu > 0$.

Given a digraph D representing an instance of MPG, we construct an instance I of (ES) in the following way. First, we assume without loss of generality that every node in D associated to the minimization player has out-degree one—it is an easy observation that a node with out-degree $q > 1$ can be replaced by q copies with out-degree one without changing the outcome of the game. Moreover, we assume that the weight of the only arc (w, j) leaving a node w associated to the minimization player is 0. The case of $d_{wj} \neq 0$ can be handled by replacing $d_{wj} \neq 0$ by 0 and d_{iw} by $d_{iw} + d_{wj}$ for all $i \in in(w)$; recall the transformation in the second paragraph of section 7.1.

The set V of jobs in the instance I of (ES) is the set of nodes associated to the maximization player. For each node w of the minimization player, we introduce a waiting condition $w = (in_D(w), j)$ where $out_D(w) = \{j\}$. The time lag d_{iw} for $i \in in_D(w)$ is given by the corresponding arc weight in D. Moreover, we add a dummy start node a preceding all other AND-nodes. We refer to this scheduling instance as I'. Finally, in order to obtain the instance I, we modify every waiting condition $w = (X, j)$ with $j \neq j_0$ and $j_0 \notin X$ by adding j_0 to X and setting $d_{j_0 w} = T + 1$; in other words, if job j_0 can be planned, then all other jobs can be planned, too. In particular, instance I is feasible if and only if the earliest start time $S^*_{j_0}$ of job j_0 in instance I' is finite. Thus, it remains to show that $S^*_{j_0} < \infty$ if and only if $\nu \leqslant 0$.

Only if: Consider instance I'. Based on the optimal partial schedule S^* of instance I' ($S_{j_0} < \infty$), we give a strategy for the minimization player which ensures $\nu \leqslant 0$: In each OR-node w with $S^*_w < \infty$, choose an incoming arc (j, w) with $S^*_j + d_{jw} = S^*_w$. As a consequence, $S^*_i < \infty$ for all nodes i visited during the game. Moreover, for two vertices v_1 and v_2 on the path formed by the game, the weight of the (directed) subpath from v_1 to v_2 is at most $S^*_{v_2} - S^*_{v_1}$. In particular, the length of the cycle terminating the game is at most 0.

If: For an infeasible scheduling instance I, by Lemma 7.3, there exists a generalized cycle C in the corresponding digraph D_I which contains only cycles of positive length. Without loss of generality, C contains the node j_0. Notice that C also forms a generalized cycle for the digraph D. We give a strategy for the maximization player which ensures $\nu > 0$: In each step, choose an arc which starts at a node in C. Such an arc always exists by the definition of generalized cycles. Moreover, again by the definition of generalized cycles, the minimization player is not able to leave C. This yields $\nu > 0$. □

With Lemma 7.5, it follows from [15] that the decision problem corresponding to the scheduling problem (ES) is in UP ∩ co-UP. Moreover, MPG and hence also (ES) can be computed in subexponential time: Zwick and Paterson [29] have shown that so-called *simple stochastic games* are at least as hard as MPGs. The outcome of simple stochastic games can be computed in subexponential time, as has been shown by Ludwig [19]. Despite these observations, there is no polynomial-time algorithm for (ES) with arbitrary arc weights known.

7.3. Positive arc weights. In this section we restrict ourselves to the case of positive arc weights or, more generally, nonnegative arc weights without cycles of length 0 in D. As in [16] and [6] we basically obtain a slight generalization of Dijkstra's shortest path algorithm. During the course of the algorithm we call a job *planned* as soon as its start time has been fixed.

The algorithm maintains a partial schedule $S \in (\mathbb{Z} \cup \{\infty\})^{|\mathcal{V}|}$ where initially $S_w = \infty$ for all OR-nodes w. All OR-nodes which are not yet planned are maintained in a heap where the sorting key for node w is its tentative start time S_w (initially $S_w = \infty$).

Having set $S_s = 0$ (and also $S_w = 0$ for all $w \in out(s)$) we proceed over time by always choosing an OR-node $w = (X, j)$ with minimum start time from the heap and plan w at its tentative start time S_w. If all other OR-nodes (X', j) preceding j have already been planned, we also plan j at the current time. In this case, the start times of all OR-nodes w' with $w' \in out(j)$ are updated to $S_{w'} := \min\{S_{w'}, S_j + d_{jw'}\}$. If after termination some OR-node w is started at $S_w = \infty$, the considered instance is infeasible. Implementational details are given in Algorithm 3.

If we apply Algorithm 3 to Example 1 (arc weights are given in Figure 2.1), we

Algorithm 3: Computation of earliest job start times for digraphs without cycles of length 0.

Input : A directed graph D representing a set V of jobs and waiting conditions $\mathcal{W}$ with positive arc weights on the arcs in $V \times \mathcal{W}$.

Output: A feasible (partial) schedule $S \in (\mathbb{Z} \cup \{\infty\})^{|\mathcal{V}|}$.

```
Heap := ∅;
for AND-nodes j ∈ V do a(j) := |in(j)|;
1 S_s := 0;    // AND-node s is planned at time 0
for OR-nodes w ∈ W do
    if w ∈ out(s) then insert w in Heap with key S_w := 0;
    else insert w in Heap with key S_w := ∞;

while Heap ≠ ∅ do
2   remove next OR-node w_0 = (X, j) from Heap;  // OR-node is planned
    reduce a(j) by 1;
    if a(j) = 0 then
3       S_j := max_{w∈in(j)} S_w; // AND-node is planned
        for OR-nodes w ∈ out(j) do
            S_w := min{S_w, S_j + d_jw};
            decrease key of w in Heap to S_w;
    delete node w_0 and all incident arcs from D;
return S;
```

obtain the start times $(0, 0, 0, 2, 3, 2, 3)$ for AND-nodes and $(2, 1, 2, 3, 3)$ for OR-nodes. One possible order in which start times get fixed is $j_1 \prec j_2 \prec j_3 \prec w_2 \prec w_1 \prec j_4 \prec w_3 \prec j_6 \prec w_5 \prec j_7 \prec w_4 \prec j_5$.

THEOREM 7.6. *For a given set of* AND/OR *precedence constraints represented by a digraph $D = (V \cup \mathcal{W}, A)$ with nonnegative arc weights and without cycles of length* 0*, Algorithm* 3 *computes an optimal partial schedule S. In particular, the instance is infeasible if and only if $S_w = \infty$ for some* OR*-node w.*

Proof. In this proof we say that an AND-node is *planned* if its start time is fixed (lines 1 and 3), while an OR-node is planned if it is removed from the heap (line 2).

By construction of Algorithm 3, S is a feasible partial schedule. Assume that S is not optimal and let v be a node with $S_v > S^*_v$ and S^*_v minimal. If v is an AND-node, then there must exist an OR-node $w = (X, v)$ with $S_w = S_v > S^*_v \geqslant S^*_w$ and we set $v := w$. Otherwise, if v is an OR-node (X, j), then there must exist an AND-node $i \in X$ with $S^*_v = S^*_i + d_{iv}$ (otherwise S^* is not minimal). Moreover, $S_i > S^*_i$. To see this suppose that $S_i = S^*_i$. Then, at the stage of Algorithm 3 where S_i is planned, v has already been removed from D. Since start times (in the order in which nodes are planned) are nondecreasing we have $S_v \leqslant S^*_i$, and hence $S_v \leqslant S_v + d_{iv} \leqslant S^*_i + d_{iv} = S^*_v$, a contradiction of $S_v > S^*_v$.

Since v was chosen such that S^*_v is minimal, we have $S^*_i \geqslant S^*_v = S^*_i + d_{iv}$. Thus $d_{iv} = 0$ and we set $v := i$. Iterating this argument, we can construct a cycle (since there are only finitely many nodes) of length 0—a contradiction. □

LEMMA 7.7. *Algorithm* 3 *can be implemented to run in* $O(|\mathcal{W}| \log |\mathcal{W}| + |A| + |V|)$ *time.*

Proof. Since each OR-node enters the heap precisely once, the while-loop is exe-

cuted $|\mathcal{W}|$ times. Each AND-node is planned only once and therefore the inner for-loop is executed at most $|A|$ times. If we choose a Fibonacci-heap for maintaining the OR-nodes, the cost of line 2 is $\log|\mathcal{W}|$ and we obtain the claimed running time. □

In contrast to previously proposed algorithms, the heap data structure maintains only OR-nodes, which leads to the improved running time $O(|\mathcal{W}|\log|\mathcal{W}|+|A|+|V|)$ instead of $O((|V|+|\mathcal{W}|)\log(|V|+|\mathcal{W}|)+|A|)$.

7.4. Nonnegative arc weights. As an extension of the case discussed in section 7.3 we present an $O(|V|+|A|\cdot|\mathcal{W}|)$ algorithm that is capable of dealing with arbitrary arc weights $d_{jw} \geqslant 0$ and thus with cycles of length 0 in D. Levner, Sung, and Vlach [18] observed that the algorithm proposed by Knuth [16] fails to compute earliest job start times when cycles of length 0 occur.

After submission of this paper for publication, we learned that Adelson-Velsky and Levner [1] also discovered a polynomial-time algorithm for the problem. Their algorithm proceeds over time starting at time $t = 0$. For each t, the algorithm determines all jobs to be started at t by an appropriate labeling procedure which runs in $O(|A|)$ time. Thereafter, t is increased to the next tentative start time of some job. It may happen, however, that for some t considered, no job is started. Adelson-Velsky and Levner [1], [2] show that the number of times t is increased is bounded by $|A|$. Consequently, they obtain an $O(|A|^2)$ algorithm. Due to a more careful update of (tentative) start times of jobs, our algorithm's worst-case running time is $O(|V|+|A|\cdot|\mathcal{W}|)$.

Adelson-Velsky and Levner [2] misleadingly claim that our algorithm solves only a special case of the problem (in which only a single arc leaves each OR-node). A digraph D with multiple arcs (w,j) leaving an OR-node w can obviously be polynomially transformed into a digraph where only a single arc leaves each OR-node as follows: Add a new AND-node i, a new arc (w,i), and add a waiting condition $(i,\{j\})$ for each arc (w,j). Then remove all arcs (w,j). With this transformation our algorithm's worst-case complexity is also $O(|A|^2)$ in the original input size. However, in Lemma 7.9 below we argue that our algorithm can also handle the case of multiple arcs leaving an OR-node directly without a transformation.

A rough scheme of the algorithm is as follows. Analogously to Algorithm 3 we maintain all OR-nodes w in a heap where the sorting key is its tentative start time S_w (initially $S_w = \infty$). Furthermore, whenever an AND-node j is planned, the start times of all OR-nodes $w \in out(j)$ are updated to $S_w = \min\{S_w, S_j + d_{jw}\}$. We proceed over time starting at $t = 0$. For the current time t we compute a set U of (nonstarted) nodes that can be started at t. The set U is computed by maintaining the induced subgraph D^0 of D where all planned nodes and all arcs of positive weight have been deleted. In D^0, the set U is computed as a set of nodes such that for each AND-node j, all predecessors $w \in in_{D^0}(j)$ are also in U, and for each OR-node w, at least one predecessor $j \in in_{D^0}(w)$ is also in U. Then, as we will prove in Theorem 7.8 below, all nodes of U can be started at the current time t. Next we remove a new OR-node w from the heap and increase t to S_w. If $t = \infty$, the algorithm stops. Then either no OR-node was left in the heap (and we have computed a feasible schedule) or all OR-nodes w in the heap fulfill $S_w = \infty$ (indicating that the given instance is infeasible). Details are provided in Algorithm 4.

If we apply Algorithm 4 to Example 1 with arc weights as in Figure 2.1 except $d_{5w_1} = 0$ and $d_{4w_4} = 0$, we get the following:

Iteration 1: $U = \{j_1, j_2, j_3\}$, $w = w_2$, $t := 1$.

Iteration 2: $U = \{j_4, j_5, w_1, w_4\}$, $w = w_3$, $t := 2$.

Algorithm 4: Computation of earliest job start times for nonnegative time lags.

Input : A directed graph D representing a set V of jobs and waiting conditions $\mathcal{W}$ with nonnegative arc weights on the arcs in $V \times \mathcal{W}$.

Output: A feasible (partial) schedule $S \in (\mathbb{Z} \cup \{\infty\})^{|\mathcal{V}|}$.

```
set D^0 := D and remove all arcs with positive weight from D^0;
t := 0;
Heap := ∅;
for OR-nodes w ∈ W do
    S_w := ∞;
    insert w in Heap with key S_w;
while t < ∞ do
    compute U ⊆ V(D^0) maximal with
1       (in_{D^0}(j) ⊆ U   ∀ j ∈ U ∩ V) and (in_{D^0}(w) ∩ U ≠ ∅   ∀ w ∈ U ∩ W);
    for AND-nodes j ∈ U  (j ∈ U ∩ V)  do
        S_j := t;        // node j is planned at time t
        for OR-nodes w ∈ out_D(j) do
2           S_w := min{S_w, S_j + d_jw};
3           decrease key of w in Heap to S_w;
    for OR-nodes w ∈ U  (w ∈ U ∩ W)  do
4       S_w := t;        // node w is planned at time t
        remove w from Heap;
    Delete all nodes from U in D and D^0;
    if Heap ≠ ∅ then
5       remove the next OR-node w from Heap;
6       t := S_w;
7       remove w from D and D^0;        // node w is planned at time t
    else  t := ∞;
return S;
```

Iteration 3: $U = \{j_6\}$, $w = w_5$, $t := 2$.
Iteration 4: $U = \{j_7\}$, $Heap = \emptyset$, $t := \infty$.

Thus, we obtain start times $(0, 0, 0, 1, 1, 2, 2)$ for AND-nodes and $(1, 1, 2, 1, 2)$ for OR-nodes.

THEOREM 7.8. *For a given set of* AND/OR *precedence constraints represented by a digraph $D = (V \cup \mathcal{W}, A)$ with nonnegative weights on the arcs, Algorithm* 4 *computes an optimal partial schedule S. In particular, the instance is infeasible if and only if $S_w = \infty$ for some* OR*-node w.*

Proof. We first prove that the variable t never decreases; i.e., the algorithm proceeds over time and tries to plan the jobs (and remove them from D and D^0) as early as possible in order of nondecreasing start times. Assume that t decreases in line 6 of the algorithm and let t_0 denote its value before the decrease. Since the OR-node w determining t was not chosen in line 5 during the last iteration of the while-loop (when t was set to t_0), its tentative start time S_w has decreased during the current iteration in lines 2 and 3. This is a contradiction of $S_j = t_0$ and $d_{jw} \geqslant 0$.

Observe that the start time S_i of any node $i \in \mathcal{V}$ is never changed after the node is planned (and thus deleted from the graphs D and D^0).

We can now prove that the partial schedule S returned by Algorithm 4 is feasible by verifying all constraints of (ES). By construction of the algorithm, for an AND-node $j \in V$, every OR-node $w \in in(j)$ either has been planned before or is planned together with j in the same iteration of the while-loop; this follows from the first property of U in line 1. Thus, the constraint in (ES) corresponding to j is fulfilled.

Consider now an arbitrary OR-node $w \in \mathcal{W}$. If w is planned as part of a subset U in line 4, it follows from the second property of U in line 1 that there is a job $j \in in(w)$ with $d_{jw} = 0$, and j is planned at the same time as w. Otherwise, if w is planned in line 7 and $S_w < \infty$, the start time S_w of w must have been decreased in some iteration of the while-loop in line 2; since the start time S_j of the node $j \in V$ causing the last decrease of S_w has not changed since then, $S_w = S_j + d_{jw}$ in the final partial schedule S. Thus, the constraint in (ES) corresponding to w is fulfilled.

Next we prove that the partial schedule S returned by Algorithm 4 is optimal. Let S^* be the optimal partial schedule and assume that there are nodes $i \in \mathcal{V}$ with $S_i^* < S_i$; we choose such an i' with minimum $S_{i'}^*$ and set $t_0 := S_{i'}^*$; let $U_0 = \{i \in \mathcal{V} \mid t_0 = S_i^* < S_i\}$. We distinguish two cases.

First case. In some iteration of Algorithm 4, t adopts the value t_0. We consider the iteration of the while-loop in which t is increased above t_0 in line 6. Let D^0 be the digraph at the beginning of the iteration and U the set computed at the start of this iteration. Then $U \cap U_0 = \emptyset$ and, by maximality of U, the set $U \cup U_0$ cannot satisfy the conditions in line 1. Since S^* is a feasible partial schedule, the first condition of line 1 is valid for $U \cup U_0$, i.e., $in_{D^0}(j) \subseteq U \cup U_0$ for all $j \in (U \cup U_0) \cap V$. Thus, the second condition is violated: there exists a node $w \in U_0 \cap \mathcal{W}$ with $in_{D^0}(w) \cap (U \cup U_0) = \emptyset$. Moreover, by optimality of S^*, there exists a node $j \in in_D(w)$ with $S_w^* = S_j^* + d_{jw}$, in particular $S_j^* \leqslant t_0$. We next show that $S_j = S_j^*$. If $S_j^* < t_0$, the claim follows from the minimality of t_0. Otherwise, observe that $j \notin U_0$ (if $j \in U_0$, we have $j \in in_{D^0}(w)$, which contradicts $in_{D^0}(w) \cap (U \cup U_0) = \emptyset$). Then with $S_j^* = t_0$ and $j \notin U_0$ it follows from the definition of U_0 that $S_j = S_j^*$. In particular, S_w has been set to $S_j + d_{jw} = S_w^*$ in line 2 after j was planned. Since S_w is never increased in Algorithm 4, we get a contradiction of $S_w > S_w^*$.

Second case. The variable t never adopts the value t_0 in Algorithm 4; in particular, $t_0 > 0$ and $U_0 = \{i \in \mathcal{V} \mid S_i^* = t_0\}$. Since S^* is optimal, decreasing all start times S_j^* for $j \in U_0$ to $t_0 - 1$ violates a constraint of (ES). Thus, there exists a node $w \in U_0 \cap \mathcal{W}$ such that $S_w^* = S_j^* + d_{jw}$ for some $j \in V$ with $d_{jw} > 0$, i.e., $S_j^* < t_0$. Therefore, $S_j = S_j^*$ and S_w has been set to $S_j + d_{jw} = S_w^*$ in line 2 after j was planned. Since S_w is never increased in Algorithm 4, we get a contradiction of $S_w > S_w^*$. ∎

The bottleneck for the running time of Algorithm 4 is the computation of the set U in each iteration of the while-loop. In fact, it turns out that the linear-time algorithm for checking feasibility of a set of AND/OR precedence constraints (for the case of positive arc weights) provides an elegant and fast solution for this problem.

LEMMA 7.9. *Given a bipartite digraph D with node set $N \cup M$ and arc set A, the (unique) maximal set $U \subseteq N \cup M$ with $in_D(w) \subseteq U$ for all $w \in U \cap N$ and $in_D(j) \cap U \neq \emptyset$ for all $j \in U \cap M$ can be computed in linear time.*

Proof. First, for U and U' fulfilling the conditions given in the lemma, their union $U \cup U'$ also fulfills those conditions. Therefore, such a unique maximal subset U exists.

We show that U can be computed by applying essentially Algorithm 1 to an appropriately constructed instance. Define the set $V = M$ of jobs and the following set $\mathcal{W}$ of waiting conditions: For each $w \in N$ and each $j \in out_D(w)$, introduce a waiting condition $(in_D(w), j)$. Notice that the input size of this instance is not necessarily

linear in the input size of the given digraph D since the set $in_D(w)$ is stored once for every $j \in out_D(w)$. We can avoid this undesired increase in the input size by storing, for each $w \in N$, the corresponding waiting conditions as $(in_D(w), out_D(w))$ with the interpretation that every job in the second set is a waiting job for the first set. Algorithm 1 can easily be adapted to handle this compactified input in linear time by replacing the for-loop starting in line 1 with

for *waiting conditions* $(X, Y) \in \mathcal{W}$ *with* $i \in X$ **do**
  for $j \in Y$ **do**
    decrease $a(j)$ by 1;
    if $a(j) = 0$ **then** add j to Q;
  remove (X, Y) from $\mathcal{W}$;

By Corollary 3.3, Algorithm 1 computes a set $L \subseteq V$ such that $V' := V \setminus L$ is a maximal subset of V with the following property: For all $j \in V'$ there exists a waiting condition $(X, j) \in \mathcal{W}$ with $X \subseteq V'$. Thus, the set

$$U = \big(\{w \in N \mid in(w) \subseteq V'\} \cup V'\big) \subseteq N \cup M$$

fulfills the conditions given in the lemma, i.e., $in_D(w) \subseteq U$ for all $w \in U \cap N$ and $in_D(j) \cap U \neq \emptyset$ for all $j \in U \cap M$. Assume that there is a bigger set $U^* \supset U$ that also fulfills these conditions. By construction of U, there exists a node $j \in M \cap (U^* \setminus U)$. Since the set $U^* \cap M$ of jobs has the property described in Corollary 3.3, we get a contradiction of the maximality of V'. □

With the help of this lemma, we can now give a bound on the running time of Algorithm 4.

COROLLARY 7.10. *Algorithm* 4 *can be implemented to run in* $\mathrm{O}(\,|\mathcal{W}| \cdot |A| + |V|\,)$ *time.*

Proof. First, all isolated AND-nodes are planned and thus removed from D^0 in the first iteration of the while-loop. Moreover, in each iteration, at least one OR-node is removed from D^0 and the number of iterations is thus bounded by $|\mathcal{W}|$. Finally, the running time of each iteration is dominated by the computation of U, which can be done in $\mathrm{O}(\,|A|\,)$ time. □

Notice that, in the sense of Lemma 7.9, Algorithm 4 and its worst-case complexity (Corollary 7.10) are both valid for digraphs D where OR-jobs have multiple outgoing arcs (of length 0).

8. Concluding remarks. The contribution of the paper is twofold. On the one hand we have provided efficient algorithms for various basic problems that occur when scheduling jobs subject to AND/OR precedence constraints (i.e., generalized feasibility, transitivity, and the computation of earliest start times of jobs for nonnegative arc weights). On the other hand we have provided further insights for solving the problem (ES) with arbitrary arc weights ($-M \leq d_{jw} \leq M$) that may help to design a polynomial-time algorithm for this important problem. In particular, the feasibility criterion (Lemma 7.3) and the algorithm for nonnegative arc weights (Algorithm 4) may be helpful.

Acknowledgments. We would like to thank Gerhard Woeginger for helpful discussions and for bringing the references [29], [15], and [24] to our attention. Furthermore, we thank Martin Zachariasen for pointing us to the work of Adelson-Velsky and Levner [1].

REFERENCES

[1] G. M. Adelson-Velsky and E. Levner, *Project Scheduling in AND/OR Graphs: A Generalization of Dijkstra's Algorithm*, Technical report, Department of Computer Science, Holon Academic Institute of Technology, Holon, Israel, 1999.

[2] G. M. Adelson-Velsky and E. Levner, *Project scheduling in AND/OR graphs: A generalization of Dijkstra's algorithm*, Math. Oper. Res., 27 (2002), pp. 504–517.

[3] G. Ausiello, A. d'Atri, and D. Saccà, *Graph algorithms for functional dependency manipulation*, J. ACM, 30 (1983), pp. 752–766.

[4] G. Ausiello, A. d'Atri, and D. Saccà, *Minimal representation of directed hypergraphs*, SIAM J. Comput., 15 (1986), pp. 418–431.

[5] F. Chauvet and J.-M. Proth, *The PERT-Problem with Alternatives: Modelisation and Optimisation*, Technical report, Institut National de Recherche en Informatique et en Automatique (INRIA), France, 1999.

[6] E. A. Dinic, *The fastest algorithm for the PERT problem with AND- and OR-nodes (the new-product-new technology problem)*, in Proceedings of the International Conference on Integer Programming and Combinatorial Optimization, R. Kannan and W. R. Pulleyblank, eds., University of Waterloo Press, Waterloo, Canada, 1990, pp. 185–187.

[7] W. F. Dowling and J. H. Gallier, *Linear-time algorithms for testing satisfiability of propositional Horn formulae*, J. Logic Program., 1 (1984), pp. 267–284.

[8] A. Ehrenfeucht and J. Mycielski, *Positional strategies for mean payoff games*, Internat. J. Game Theory, 8 (1979), pp. 109–113.

[9] G. Gallo, C. Gentile, D. Pretolani, and G. Rago, *Max Horn SAT and the minimum cut problem in directed hypergraphs*, Math. Programming, 80 (1998), pp. 213–237.

[10] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen, *Directed hypergraphs and applications*, Discrete Appl. Math., 42 (1993), pp. 177–201.

[11] D. W. Gillies, *Algorithms to Schedule Tasks with AND/OR Precedence Constraints*, Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1993.

[12] D. W. Gillies and J. W.-S. Liu, *Scheduling tasks with AND/OR precedence constraints*, SIAM J. Comput., 24 (1995), pp. 797–810.

[13] M. H. Goldwasser and R. Motwani, *Complexity measures for assembly sequences*, Internat. J. Comput. Geom. Appl., 9 (1999), pp. 371–418.

[14] G. Igelmund and F. J. Radermacher, *Algorithmic approaches to preselective strategies for stochastic scheduling problems*, Networks, 13 (1983), pp. 29–48.

[15] M. Jurdziński, *Deciding the winner in parity games is in $UP \cap co\text{-}UP$*, Inform. Process. Lett., 68 (1998), pp. 119–124.

[16] D. E. Knuth, *A generalization of Dijkstra's algorithm*, Inform. Process. Lett., 6 (1977), pp. 1–5.

[17] B. Korte, L. Lovász, and R. Schrader, *Greedoids*, Springer-Verlag, Berlin, 1991.

[18] E. Levner, S. C. Sung, and M. Vlach, *Makespan minimization in projects with threshold activities*, Asia-Pacific J. Oper. Res., 19 (2002), pp. 195–204.

[19] W. Ludwig, *A subexponential randomized algorithm for the simple stochastic game problem*, Inform. and Comput., 117 (1995), pp. 151–155.

[20] K. L. McMillan and D. L. Dill, *Algorithms for interface timing verification*, in Proceedings of the IEEE International Conference on Computer Design, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 48–51.

[21] R. H. Möhring, M. Skutella, and F. Stork, *Forcing relations for AND/OR precedence constraints*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00), ACM, New York, SIAM, Philadelphia, 2000, pp. 235–236.

[22] R. H. Möhring and F. Stork, *Linear preselective policies for stochastic project scheduling*, Math. Methods Oper. Res., 52 (2000), pp. 501–515.

[23] N. J. Nilsson, *Principals of Artificial Intelligence*, Tioga, Palo Alto, CA, 1980.

[24] U. Schwiegelshohn and L. Thiele, *Dynamic min-max problems*, Discrete Event Dyn. Syst., 9 (1999), pp. 111–134.

[25] C. Schwindt, *A Branch-and-Bound Algorithm for the Resource-Constrained Project Duration Problem Subject to Temporal Constraints*, Technical report 544, WIOR, University of Karlsruhe, Germany, 1998.

[26] F. Stork, *Stochastic Resource-Constrained Project Scheduling*, Ph.D. thesis, Department of Mathematics, Technische Universität Berlin, 2001.

[27] J. D. Ullman, *Principles of Database Systems*, 2nd ed., Computer Science Press, Rockville, MD, 1982.

[28] J. Vöge and M. Jurdziński, *A discrete strategy improvement algorithm for solving parity games*, in Proceedings of the 12th International Conference of Computer Aided Verification, CAV 2000, E. A. Emerson and A. P. Sistla, eds., Lecture Notes in Comput. Sci. 1855, Springer-Verlag, Chicago, IL, 2000, pp. 202–215.

[29] U. Zwick and M. Paterson, *The complexity of mean payoff games on graphs*, Theoret. Comput. Sci., 158 (1996), pp. 343–359.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 416–432

THE COMPLEXITY OF CHOOSING AN H-COLORING (NEARLY) UNIFORMLY AT RANDOM*

LESLIE ANN GOLDBERG†, STEVEN KELK†, AND MIKE PATERSON†

Abstract. Cooper, Dyer, and Frieze [*J. Algorithms*, 39 (2001), pp. 117–134] studied the problem of sampling H-colorings (nearly) uniformly at random. Special cases of this problem include sampling colorings and independent sets and sampling from statistical physics models such as the Widom–Rowlinson model, the Beach model, the Potts model and the hard-core lattice gas model. Cooper et al. considered the family of "cautious" ergodic Markov chains with uniform stationary distribution and showed that, for every fixed connected "nontrivial" graph H, every such chain mixes slowly. In this paper, we give a complexity result for the problem. Namely, we show that for *any* fixed graph H with no trivial components, there is unlikely to be any *polynomial almost uniform sampler* (PAUS) for H-colorings. We show that if there were a PAUS for the H-coloring problem, there would also be a PAUS for sampling independent sets in bipartite graphs, and, by the self-reducibility of the latter problem, there would be a *fully polynomial randomized approximation scheme* (FPRAS) for #BIS—the problem of counting independent sets in bipartite graphs. Dyer, Goldberg, Greenhill, and Jerrum have shown that #BIS is complete in a certain logically defined complexity class. Thus, a PAUS for sampling H-colorings would give an FPRAS for the entire complexity class. In order to achieve our result we introduce the new notion of *sampling-preserving* reduction which seems to be more useful in certain settings than approximation-preserving reduction.

Key words. graph coloring, graph homomorphism, computational complexity, random sampling

AMS subject classifications. 68W20, 05C15, 68R10

DOI. 10.1137/S0097539702408363

1. Introduction.

Let $H = (V(H), E(H))$ be any fixed graph. An *H-coloring* of a graph $G = (V(G), E(G))$ is just a homomorphism from G to H: The vertices of H correspond to "colors," and the edges of H specify which colors may be adjacent. Thus, an H-coloring of G is a function C from $V(G)$ to $V(H)$ such that for every edge $(u, v) \in E(G)$ the corresponding edge $(C(u), C(v))$ is in $E(H)$. Informally, colors $C(u)$ and $C(v)$ are allowed to be adjacent in the coloring C of G because the edge $(C(u), C(v))$ is an edge of H.

Many combinatorial problems can be viewed as special cases of H-coloring. For example, if H is a k-clique with no self-loops, then H-colorings of G correspond to proper k-colorings of G. (In such a coloring, k colors are available for coloring the vertices of G, but no color may be adjacent to itself.) Here is another example. If H is the graph depicted in Figure 1, then H-colorings of G correspond to independent sets of G—vertices which are colored "a" are in the independent set, and vertices which are colored "b" are not. Several models from statistical physics are special cases of H-coloring, including the Widom–Rowlinson model, the Beach model, and (for weighted H-colorings) the Potts model and the hard-core lattice gas model. See [2, 10] for details.

The complexity of H-coloring has been well studied. Many papers considered the

*Received by the editors May 28, 2002; accepted for publication (in revised form) September 19, 2003; published electronically February 18, 2004. This work was partially supported by EPSRC grants GR/R44560/01 "Analysing Markov-chain based random sampling algorithms" and GR/M96940 "Sharper Analysis of Randomised Algorithms: a Computational Approach" and the IST Programme of the EU under contracts IST-1999-14036 (RAND-APX) and IST-1999-14186 (ALCOM-FT).

http://www.siam.org/journals/sicomp/33-2/40836.html

†Department of Computer Science, University of Warwick, Coventry, CV4 7AL, United Kingdom.

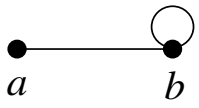


FIG. 1. *Homomorphisms from G to this graph are independent sets of G.*

following problem: Given a fixed graph H, determine, for an input graph G, whether G has an H-coloring. Hell and Nešetřil [14] completely characterized the set of graphs for which this problem is NP-complete. They observed that the problem is in P if H has a loop or is bipartite, and they showed that it is NP-complete for any other fixed H. See [14] for references to earlier work on this question and [13] for extensions to the case in which the maximum degree of G is bounded. See [4, 5] for extensions to parameterized complexity.

Dyer and Greenhill [10] considered the problem of *counting* H-colorings. Intriguingly, they were able to completely characterize the graphs H for which this problem is #P-complete. A connected component of H is said to be "trivial" if it is a complete graph with all loops present or a complete bipartite graph with no loops present.[1] Dyer and Greenhill showed that counting H-colorings is #P-complete if H has a nontrivial component and that it is in P otherwise. They also extended their result to the case in which the maximum degree of G is bounded.

Other work has focused on the complexity of *sampling* H-colorings (nearly) uniformly at random.[2] Positive results for particular graphs H (specifically for the case in which H-colorings are independent sets and for the case in which H-colorings are proper colorings) appear in works such as [9, 15, 18]. A negative result for the independent-set case appears in [6]. The first paper to study the complexity of sampling H-colorings in the general case was Cooper, Dyer, and Frieze [3]. They focused on connected graphs H for which the decision problem "Is there an H-coloring?" is in P, but the counting problem "How many H-colorings are there?" is #P-complete. They showed that for any such H, H-colorings cannot be sampled efficiently using "cautious" Markov chains, which are Markov chains which can change only a constant fraction of the colors of the vertices in a single step. In particular, the mixing time of all such chains is exponential in the number of vertices of G. They also give positive results for certain weighted cases, which are extended in [12]. In particular, [12] shows that for every fixed "dismantleable" H and every degree bound Δ there are positive vertex weights which can be assigned to the vertices of H so that weighted H-colorings can be sampled for degree-Δ graphs G. Borgs et al. [1] consider the problem of sampling H-colorings on rectangular subsets of the hypercubic lattice. They show that for every nontrivial connected H there is an assignment of weights to colors for which cautious chains are slowly mixing.

In this work, we study the complexity of sampling H-colorings. We show that if H has no trivial components, then the problem of nearly uniformly sampling H-colorings is intractable in a complexity-theoretic sense. In particular, we show that for

[1] Following the usual notation in the area, we will treat self-loops specially, so it makes sense to refer to bipartite graphs with or without loops. The loop-free single vertex is viewed as a complete bipartite graph.

[2] Some of this work has been motivated by the well-known connection between almost uniform sampling and approximate counting [17, 8]. For some graphs H, it can be shown that the problem of approximately counting H-colorings is equivalent to the problem of sampling H-colorings (nearly) uniformly at random. See section 8.

any fixed H with no trivial components there is unlikely to be any *polynomial almost uniform sampler* (PAUS) for H-colorings. We show that if there were a PAUS for the H-coloring problem, there would also be a PAUS for sampling independent sets in bipartite graphs, and, by the self-reducibility of the latter problem, there would be a *fully polynomial randomized approximation scheme* (FPRAS) for #BIS—the problem of counting independent sets in bipartite graphs. Dyer, Goldberg, Greenhill and Jerrum have shown that #BIS is complete in a certain logically defined subclass of #P which includes problems such as counting downsets in partial orders and counting satisfying assignments in "restricted Horn" CNF Boolean formulas. Thus, a PAUS for sampling H-colorings would give an FPRAS for the entire complexity class. In fact, our result holds even if the input G is restricted to be a connected bipartite graph.

In order to achieve our result we introduce the new notion of *sampling-preserving* reduction. The notion of approximation-preserving reduction (AP-reducibility) from [11] seems to be too demanding. In particular, since AP-reducibility is about *counting* (as opposed to sampling), an AP-reduction is not allowed to inflate the size of the set of structures by a factor which is difficult to compute. Sampling-preserving reductions allow this flexibility while achieving the same final result. The definition of sampling reduction (section 2) is essentially many-one. Nevertheless, the reductions get used in a "Turing reduction" way. In particular, our reduction from SampleBIS to Sample$H$-Col takes an instance of SampleBIS and constructs many Sample$H$-Col instances. Since the resulting maps between H-colorings and independent sets are many-one, several reductions can be combined even though they may involve different amounts of inflation of the state space.

The paper is structured as follows. Section 2 gives the relevant definitions including the definition of a sampling-preserving reduction. Section 3 presents some technical lemmas which we need in our proofs. Section 4 outlines a general proof technique for demonstrating the existence of an SP-reduction. Section 5 uses the new proof technique to reduce SampleBIS to a crucial intermediate problem, SampleFixed$H$-Col. Section 6 proves the main result. Sections 7 and 8 discuss extensions.

2. Definitions. The total variation distance between two distributions π and π' on a countable set Ω is given by

$$\mathrm{d_{TV}}(\pi, \pi') = \frac{1}{2} \sum_{\omega \in \Omega} |\pi(\omega) - \pi'(\omega)| = \max_{A \subseteq \omega} |\pi(A) - \pi'(A)|.$$

A sampling problem X maps each instance σ to a set of structures $X(\sigma)$. The goal is to produce a member of $X(\sigma)$ uniformly at random. The size of each structure in $X(\sigma)$ is at most a polynomial in $|\,\sigma\,|$. For a given graph H, the sampling problem Sample$H$-Col will be defined as follows.

Name. Sample$H$-Col.
Instance. A loop-free graph G.
Output. An H-coloring of G chosen uniformly at random.

We will be particularly interested in the special case of this problem in which the input graph, G, is connected and bipartite.

Name. SampleBH-Col.
Instance. A loop-free connected bipartite graph G.
Output. An H-coloring of G chosen uniformly at random.

The problem SampleBIS will be defined as follows.

Name. SAMPLEBIS.
Instance. A loop-free connected bipartite graph G.
Output. An independent set of G chosen uniformly at random.

An *almost uniform sampler* [8, 16, 17] for X is a randomized algorithm that takes input σ and accuracy parameter $\epsilon \in (0, 1]$ and gives an output such that the variation distance between the output distribution of the algorithm and the uniform distribution on $X(\sigma)$ is at most ϵ. We will say that algorithm is a *polynomial almost uniform sampler (PAUS)* if its running time is bounded from above by a polynomial in the size of the instance $|\sigma|$ and $1/\epsilon$.

A *sampling-preserving reduction* (SP-reduction) from a sampling problem X to a sampling problem Y (denoted $X \leq_{\mathrm{SP}} Y$) consists of the following:

1. A function f which takes an input (σ, ϵ), in which σ is an instance of X and $\epsilon \in (0, 1]$ is an accuracy parameter, and produces an instance $f(\sigma, \epsilon)$ of Y. If $X(\sigma)$ is nonempty, then $Y(f(\sigma, \epsilon))$ must be nonempty.
2. A function g which maps each tuple (σ, ϵ, y) with $y \in Y(f(\sigma, \epsilon))$ to a member of $X(\sigma) \cup \{\perp\}$, where "$\perp$" is an error symbol and for every (σ, ϵ) and every $x \in X(\sigma)$,

$$(1) \quad e^{-\epsilon}\frac{|Y(f(\sigma,\epsilon))|}{|X(\sigma)|} \leq |\{y \in Y(f(\sigma,\epsilon)) \mid g(\sigma,\epsilon,y) = x\}| \leq e^{\epsilon}\frac{|Y(f(\sigma,\epsilon))|}{|X(\sigma)|}.$$

Equation (1) says that for every $x \in X(\sigma)$ the number of $y \in Y(f(\sigma, \epsilon))$ which are mapped to x by g is roughly $\frac{|Y(f(\sigma,\epsilon))|}{|X(\sigma)|}$. Thus, each $x \in X(\sigma)$ is roughly equally represented, and the error symbol $\perp$ is represented by only about an ϵ-fraction of $Y(f(\sigma, \epsilon))$.

The functions f and g must be computable in time which is bounded by a polynomial in $|\sigma|$ and $1/\epsilon$.

The motivation for this definition is the following lemma.

LEMMA 1. *If $X \leq_{\mathrm{SP}} Y$ and Y has a PAUS, then X has a PAUS.*

Proof. Let (f, g) be the reduction from X to Y, and let $\mathcal{A}$ be a PAUS for Y. Here is a PAUS for X: On input (σ, ϵ), let y be the output of $\mathcal{A}$ when it is run with inputs $f(\sigma, \epsilon/4)$ and $\epsilon/2$; return $g(\sigma, \epsilon/4, y)$. We must show that the variation distance between the output distribution of this algorithm and the uniform distribution on $X(\sigma)$ is at most ϵ. Let σ be an input with $|X(\sigma)| \geq 1$. Consider any subset A_x of $X(\sigma)$. Let

$$A_y = \{y \in Y(f(\sigma, \epsilon/4)) \mid g(\sigma, \epsilon/4, y) \in A_x\}.$$

Then the probability that $\mathcal{A}$ gives an output in A_y is at most

$$\begin{aligned}
&\frac{|A_y|}{|Y(f(\sigma,\epsilon/4))|} + \frac{\epsilon}{2} \\
&\leq \frac{e^{\epsilon/4}|A_x|}{|X(\sigma)|} + \frac{\epsilon}{2} \\
&\leq \frac{(1+\epsilon/2)|A_x|}{|X(\sigma)|} + \frac{\epsilon}{2} \\
&\leq \frac{|A_x|}{|X(\sigma)|} + \frac{(\epsilon/2)|A_x|}{|X(\sigma)|} + \frac{\epsilon}{2} \\
&\leq \frac{|A_x|}{|X(\sigma)|} + \epsilon.
\end{aligned}$$

Also, the probability that $\mathcal{A}$ gives an output in A_y is at least

$$\begin{aligned}
&\frac{|A_y|}{|Y(f(\sigma,\epsilon/4))|} - \frac{\epsilon}{2} \\
&\geq \frac{e^{-\epsilon/4}|A_x|}{|X(\sigma)|} - \frac{\epsilon}{2} \\
&\geq \frac{(1-\epsilon/2)|A_x|}{|X(\sigma)|} - \frac{\epsilon}{2} \\
&\geq \frac{|A_x|}{|X(\sigma)|} - \frac{|A_x|(\epsilon/2)}{|X(\sigma)|} - \frac{\epsilon}{2} \\
&\geq \frac{|A_x|}{|X(\sigma)|} - \epsilon. \qquad \square
\end{aligned}$$

The problem #BIS is defined as follows.
Name. #BIS.
Instance. A loop-free bipartite graph G.
Output. The number of independent sets of G.

A component of H is *trivial* if it is a complete graph with all loops present or a complete bipartite graph with no loops present. Recall from Dyer and Greenhill [10] that counting H-colorings is in P if H is trivial. The main result of this paper is as follows.

THEOREM 2. *Suppose that H is a fixed graph with no trivial components. If* SAMPLEBH-COL *has a PAUS then* SAMPLEBIS *has a PAUS, and* #BIS *has an FPRAS. Thus, every problem which is AP-interreducible with* #BIS *(see* [11]*) has an FPRAS.*

3. Technical lemmas. Let $\nu(a,b)$ denote the number of onto functions from a set of size a to a set of size b. We need to use the following lemma, which is Lemma 18 of [11].

LEMMA 3 (DGGJ). *If a and b are positive integers and $a \geq 2b\ln b$, then*

$$b^a\left(1-\exp(-a/(2b))\right) \leq \nu(a,b) \leq b^a.$$

We also need the following technical lemma.

LEMMA 4. *Suppose c_1 and c_2 are fixed positive reals with $c_1 < c_2$. For any $\delta > 0$ and any nonnegative integers q and a_0, there are nonnegative integers a and b with $a \geq a_0$ which are in $O((a_0+q)/\delta)$ and satisfy*

$$e^{-\delta}c_2^{a+q} \leq c_1^{b+q} \leq e^{\delta}c_2^{a+q}.$$

Proof. First, note that it would suffice to find nonnegative integers a' and b' which are in $O(q'/\delta)$ and satisfy

$$e^{-\delta}c_2^{a'+q'} \leq c_1^{b'+q'} \leq e^{\delta}c_2^{a'+q'},$$

where $q' = q + a_0$ because we could simply set $a = a' + a_0$ and $b = b' + a_0$ which would imply $a' + q' = a + q$ and $b' + q' = b + q$.

Taking logarithms, what we need is

$$\left| b' - \frac{a'\log c_2 + q'\log(c_2/c_1)}{\log c_1} \right| \leq \frac{\delta}{\log c_1}. \tag{2}$$

Now let ρ be defined by $c_2 = c_1^{1+\rho}$. Then we want

$$|b' - (a'(1+\rho) + q'\rho)| \leq \frac{\delta}{\log c_1}. \tag{3}$$

For a positive integer r, we will choose $a' = q'r$, so we want

$$|b' - a' - \rho q'(r+1)| \leq \frac{\delta}{\log c_1}. \tag{4}$$

Let $R = \lceil 2 \log c_1/\delta \rceil$. Lemma 19 of [11] says the following: *For any real $z > 0$ and any positive integer R there is an $x \in [1, \ldots, R]$ such that*

$$\min(zx - \lfloor zx \rfloor, \lceil zx \rceil - zx) \leq 1/R.$$

Thus, there is an $x \in [1, \ldots, R]$ such that $\rho q' x$ is within $1/R$ of a nonnegative integer. If $x > 1$ we will set $r + 1 = x$. If $x = 1$, then note that $\rho q' 2$ is within $2/R$ of a nonnegative integer, so we will set $r = 1$.

Now recall that $a' = q'r$, so $a' \in O(q'/\delta)$ as required. □

4. Demonstrating the existence of SP-reductions: A proof technique. When we introduce an SP-reduction from a sampling problem X to a sampling problem Y, we will need to show that (1) is satisfied. We will typically do this by partitioning $Y(f(\sigma, \epsilon))$ into disjoint sets $Y_0, \ldots, Y_k$. We will show that each of $Y_1, \ldots, Y_k$ is fairly representative of $X(\sigma)$. In particular, for every $x \in X(\sigma)$ and every $i \in [1, k]$,

$$e^{-\epsilon/2} \frac{|Y_i|}{|X(\sigma)|} \leq |\{y \in Y_i \mid g(\sigma, \epsilon, y) = x\}| \leq e^{\epsilon/2} \frac{|Y_i|}{|X(\sigma)|}. \tag{5}$$

For every $y \in Y_0$, we will have $g(\sigma, \epsilon, y) = \perp$, but we will show that Y_0 is a small part of $Y(f(\sigma, \epsilon))$. In particular,

$$\sum_{i=1}^{k} |Y_i| \geq e^{-\epsilon/2} |Y(f(\sigma, \epsilon))|. \tag{6}$$

Together, (5) and (6) imply (1). Note that (6) follows from

$$|Y_0| \leq (\epsilon/4)|Y(f(\sigma, \epsilon))|, \tag{7}$$

since (7) implies $|Y| - |Y_0| \geq (1 - \epsilon/4)|Y(f(\sigma, \epsilon))| \geq e^{-\epsilon/2}|Y(f(\sigma, \epsilon))|$.

Quite often the reduction $X \leq_{\mathrm{SP}} Y$ will involve several subproblems $Z_1, Z_2, \ldots$ such that, for each of these, an SP-reduction (f_i, g_i) from X to Z_i is already known. The instance $f(\sigma, \epsilon)$ of Y is then formed by "gluing" together instances $f_1(\sigma, \epsilon/2)$ of Z_1, $f_2(\sigma, \epsilon/2)$ of Z_2, and so on. Y_i is (roughly) the portion of $Y(f(\sigma, \epsilon))$ for which, within each $y \in Y_i$, we can "zoom in" on a structure $z \in Z_i(f_i(\sigma, \epsilon/2))$. Each structure in $Z_i(f_i(\sigma, \epsilon/2))$ is represented by an equal number of $y \in Y_i$ so we can get (5) by referring to the SP-reduction from X to Z_i. Establishing (7) is essentially showing that, although $Y(f(\sigma, \epsilon))$ has some structures which do not allow us to "zoom in" on an appropriate subproblem to find our sample, these are not so numerous.

Finally, let $Y_i(x) = \{y \in Y_i \mid g(\sigma, \epsilon, y) = x\}$. Suppose that no $y \in Y_i$ has $g(\sigma, \epsilon, y) = \perp$. In this case we can show (5) by showing that for all $x, x' \in X(\sigma)$,

$$|Y_i(x)| \leq e^{\epsilon/2} |Y_i(x')|. \tag{8}$$

To see this, note that

$$\frac{|Y_i|}{|X(\sigma)|} = \frac{\sum_{x' \in X(\sigma)} |Y_i(x')|}{|X(\sigma)|} \geq e^{-\epsilon/2} \frac{\sum_{x' \in X(\sigma)} |Y_i(x)|}{|X(\sigma)|} = e^{-\epsilon/2} |Y_i(x)|.$$

5. Sampling *fixed H*-colorings. Suppose that H is connected, loop-free, and bipartite. Denote the vertex partition of H by $(V_L(H), V_R(H))$. We will define the *fixed* H-coloring problem as follows.

Name. SampleFixed$H$-Col.

Instance. A loop-free connected bipartite graph G with vertex partition $(V_L(G), V_R(G))$.

Output. An H-coloring of G chosen uniformly at random from the set of H-colorings in which vertices of $V_L(G)$ receive colors from $V_L(H)$.

We will study the problem SampleFixed$H$-Col as an intermediate step on the way to the proof of Theorem 2.

A vertex in $V_L(H)$ is said to be *full* if it is adjacent to every vertex in $V_R(H)$. Similarly, a vertex in $V_R(H)$ is said to be *full* if it is adjacent to every vertex in $V_L(H)$. The graph H is said to be *full* if both $V_L(H)$ and $V_R(H)$ contain at least one full vertex. The following lemma is the key ingredient in the proof of Theorem 2.

Lemma 5. *Suppose that H is a connected nontrivial full loop-free bipartite graph. Then* SampleBIS $\leq_{\text{SP}}$ SampleFixed$H$-Col.

Proof. We will prove the lemma by induction on the number of vertices in H. For the base case, suppose that H has at most 4 vertices. The only connected nontrivial full loop-free bipartite graph H with at most 4 vertices is the path of length 3. Let G be an input to SampleBIS. There is a one-to-one correspondence between independent sets of G and fixed H-colorings of G: The endpoints of H point out the vertices which are in the independent set (see the proof of Theorem 5 of [11]).

We will now move on to the inductive step. The high-level idea is the following. By considering the graph H, we will construct several graphs $H_{S_1}, \ldots, H_{S_{j+k}}$, each of which is smaller than H and satisfies certain conditions. By induction, for each i, there is an SP-reduction from SampleBIS to SampleFixedH_{S_i}-Col. If we apply this reduction to our instance G of SampleBIS, we get an instance G_i of SampleFixedH_{S_i}-Col. Our goal is to construct an instance $f(G, \epsilon)$ of SampleFixed$H$-Col. We do this by "gluing together" the various G_i's. Now consider the constructed instance $f(G, \epsilon)$ of SampleFixed$H$-Col. When we sample from the output distribution SampleFixed$H$-Col$(f(G, \epsilon))$, we would like to recover the output distribution of $SampleBIS(G)$. Curiously, we cannot determine during the reduction itself the relative weights of the subinstances $G_1, G_2, \ldots$. Nevertheless, once we have an output to SampleFixed$H$-Col$(f(G, \epsilon))$, the output itself tells us which H_i is relevant. From this, we can recover an output to SampleFixedH_{S_i}-Col(G_i) and from this we can recover an output to SampleBIS(G). The main technical difficulty lies in showing that the distributions are correct. In particular, since the subreductions are SP-reductions (i.e., the equations in section 4 are satisfied in the construction of $G_1, G_2, \ldots$), the combined reduction is also an SP-reduction.

We now describe the details. Let F_L be the set of full vertices in $V_L(H)$, and let F_R be the set of full vertices in $V_R(H)$. Let $f_L = |F_L|$ and $f_R = |F_R|$ and $v_L = |V_L(H)|$ and $v_R = |V_R(H)|$. For a subset S of $V_R(H)$, let $N(S)$ be the set of mutual neighbors of S:

$$N(S) = \{v \in V_L(H) \mid \forall u \in S, (u, v) \in E(H)\}.$$

Note that $F_L \subseteq N(S) \subseteq V_L(H)$. S is said to be *left-reducing* if $F_L \subset N(S) \subset V_L(H)$. If S is left-reducing, let H_S be the subgraph of H induced by vertex partition $(N(S), V_R(H))$. Note that H_S has fewer vertices than H. Also, it is connected, full, and nontrivial: The set of full vertices in $V_L(H_S)$ is F_L; the set of full vertices in $V_R(H_S)$ includes all of F_R, but it does not equal $V_R(H)$ since $N(S) \supset F_L$.

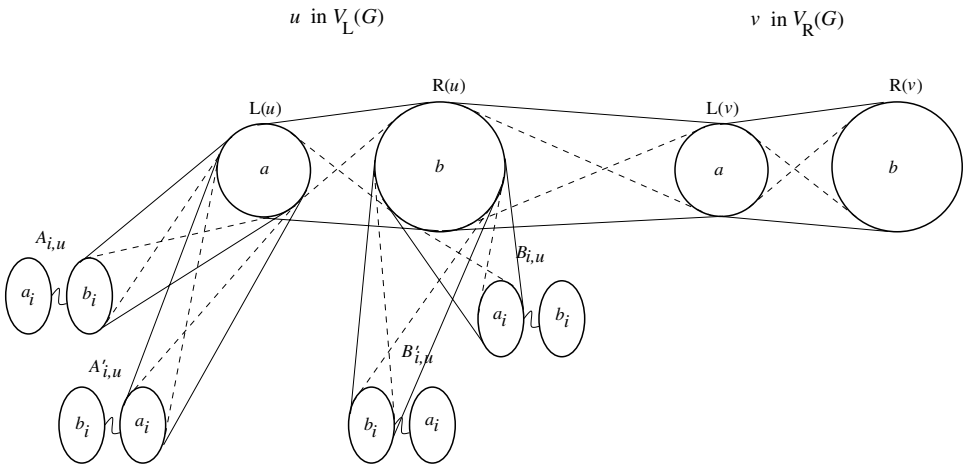


FIG. 2. *The construction of $f(G,\epsilon)$ in the proof of Lemma 5.*

Similarly, a subset S of $V_L(H)$ is *right-reducing* if $F_R \subset N(S) \subset V_R(H)$. If S is right-reducing, let H_S be the subgraph of H induced by vertex partition $(V_L(H), N(S))$. H_S has fewer vertices than H and is connected, full, and nontrivial.

Now, let $S_1, \ldots, S_k$ be the left-reducing subsets of $V_R(H)$, and let $S_{k+1}, \ldots, S_{k+j}$ be the right-reducing subsets of $V_L(H)$. (Either or both of k and j could be zero.) For every $i \in \{1, \ldots, k+j\}$, let (f_i, g_i) be an SP-reduction from SAMPLEBIS to SAMPLEFIXEDH_{S_i}-COL. Take the input (G,ϵ) to SAMPLEBIS, and let $G_i = f_i(G, \epsilon/2)$. Let $a_i = |V_L(G_i)|$, and let $b_i = |V_R(G_i)|$. Let $q = \sum_{i=1}^{k+j}(a_i + b_i)$, and let $n = |V_L(G)| + |V_R(G)|$.

Let $f(G,\epsilon)$ be the graph which is constructed as follows, where a and b will be chosen later to satisfy

$$a \geq 2v_L \lceil q \ln(v_R/f_R) + \ln(16n/\epsilon) \rceil \tag{9}$$

and

$$b \geq 2v_R \lceil q \ln(v_L/f_L) + \ln(16n/\epsilon) \rceil . \tag{10}$$

See Figure 2. For every vertex u of G, put a size-a set $L(u)$ into $V_L(f(G,\epsilon))$ and a size-b set $R(u)$ into $V_R(f(G,\epsilon))$. Also, add edges $L(u) \times R(u)$ to $E(f(G,\epsilon))$. If $u \in V_L(G)$ is connected to $v \in V_R(G)$ by an edge of G, then add edges $R(u) \times L(v)$ to $E(f(G,\epsilon))$.

Also, for every vertex u of G and every $i \in [1, \ldots, k+j]$, let $A_{i,u}$ and $B_{i,u}$ be copies of G_i, and let $A'_{i,u}$ and $B'_{i,u}$ be copies of G_i in which left vertices and right vertices are switched (so the vertices in $V_L(A'_{i,u})$ correspond to the vertices in $V_R(G_i)$ and the vertices in $V_R(A'_{i,u})$ correspond to the vertices in $V_L(G_i)$). Add edges $L(u) \times V_R(A_{i,u})$ and $L(u) \times V_R(A'_{i,u})$ and $R(u) \times V_L(B_{i,u})$ and $R(u) \times V_L(B'_{i,u})$ to $E(f(G,\epsilon))$.

Let

$$V_L(f(G,\epsilon)) = \bigcup_u L(u) \cup \bigcup_{u,i} \{V_L(A_{i,u}) \cup V_L(A'_{i,u}) \cup V_L(B_{i,u}) \cup V_L(B'_{i,u})\},$$

and let Y be the set of fixed H-colorings of $f(G,\epsilon)$. We will partition Y into sets $Y_0, \ldots, Y_{k+j+1}$.

For $i \in [1, \ldots, k]$, Y_i is the set of colorings which are not in $Y_1, \ldots, Y_{i-1}$ but in which some $u \in V_L(G)$ has $R(u)$ colored with (exactly) the colors in S_i. For $i \in [k+1, \ldots, k+j]$, Y_i is the set of colorings which are not in $Y_1, \ldots, Y_{i-1}$ but in which some $v \in V_R(G)$ has $L(v)$ colored with S_i.

The high-level structure of our construction is as follows. For every $i \in \{1, \ldots, k+j\}$, we will use the colorings in Y_i by focusing on the induced colorings of the subgraph G_i. These are H_{S_i}-colorings of G_i and from these we can (by induction) recover a random independent set of G. As usual, the colorings in Y_0 are not useful for pointing out independent sets, but there are not too many of these. Every coloring in Y_{k+j+1} has a special form—every vertex u of G has either $R(u)$ colored $V_R(H)$ or has $L(u)$ colored $V_L(H)$. These colorings point out independent sets of G in a natural way, and each independent set comes up about the same number of times in this way.

We now look at the details. Note that for any colorings in Y_0 or Y_{k+j+1} we have the following property—every vertex $u \in V_L(G)$ has $R(u)$ colored with a set S of colors such that $N(S)$ is either F_L or $V_L(H)$. Similarly, every vertex $v \in V_R(G)$ has $L(v)$ colored with a set S of colors such that $N(S)$ is either F_R or $V_R(H)$.

Consider a coloring y. Vertex $u \in V_L(G)$ satisfies *Condition* (A) if $R(u)$ is colored with a set S of colors with $N(S) = F_L$ but $S \subset V_R(H)$. It satisfies *Condition* (B) if $R(u)$ is colored with a set S of colors with $N(S) = V_L(H)$ but $L(u)$ is colored with a proper subset of $V_L(H)$. Vertex $v \in V_R(G)$ satisfies *Condition* (C) if $L(v)$ colored with a set S of colors with $N(S) = F_R$ but $S \subset V_L(H)$. It satisfies *Condition* (D) if $L(v)$ is colored with a set S of colors with $N(S) = V_R(H)$ but $R(v)$ is colored with a proper subset of $V_R(H)$.

We now define

$$Y_0 = \{y \in Y - \{Y_1 \cup \cdots \cup Y_{k+j}\} \mid \text{some vertex satisfies Condition (A), (B), (C), or (D)}\}.$$

Now note that colorings in Y_{k+j+1} have the following property. Every vertex u of G has either $R(u)$ colored $V_R(H)$ or has $L(u)$ colored $V_L(H)$.

We will first work on establishing (7). Let $Y_{u,A}$ denote the subset of Y in which u satisfies (A). Define $Y_{u,B}$, $Y_{u,C}$, and $Y_{u,D}$ similarly. We will show that the size of each of $Y_{u,A}$, $Y_{u,B}$, $Y_{u,C}$, and $Y_{u,D}$ is at most $(\epsilon/(16n))|Y|$. Equation (7) follows since

$$|Y_0| \leq \sum_{u \in V(G)} |Y_{u,A}| + |Y_{u,B}| + |Y_{u,C}| + |Y_{u,D}|.$$

First, let us show that $|Y_{u,A}| \leq (\epsilon/(16n))|Y|$. Consider the set of colorings in Y in which all neighbors of vertices in $R(u)$ have colors from F_L, and let ψ be the number of induced colorings on vertices other than the vertices of $R(u)$. If $\psi = 0$, then $|Y_{u,A}| = 0$, so the claim is trivial. Otherwise, $|Y_{u,A}| \leq \psi(v_R^b - \nu(b, v_R))$ which is at most $\psi v_R^b \exp(-b/(2v_R))$ by Lemma 3. On the other hand, $|Y| \geq \psi v_R^b$, so the claim follows from (10). The proof that $|Y_{u,C}|$ is sufficiently small is similar.

Next, let us show that $|Y_{u,B}| \leq (\epsilon/(16n))|Y|$. Consider the set of colorings in Y in which $R(u)$ is colored with a subset of F_R, and let ψ be the number of induced colorings on all vertices except those in $L(u)$ and $A_{i,u}$ and $A'_{i,u}$ (for $i \in [1, \ldots, j+k]$). If $\psi = 0$, then $|Y_{u,B}| = 0$, so the claim is trivial. Otherwise, $|Y_{u,B}| \leq \psi(v_L^a - \nu(a, v_L))v_R^q v_L^q$ which is at most $\psi v_L^a \exp(-a/(2v_L))v_R^q v_L^q$ by Lemma 3. On the other hand, $|Y| \geq \psi v_L^a f_R^q v_L^q$, so the claim follows from (9). The proof that $|Y_{u,D}|$ is sufficiently small is similar.

We will now work on establishing (5). First consider $i \in [1, \ldots, k]$. Let $Y_{u,i}$ be the set of colorings in Y_i for which $u \in V_L(G)$ is the first vertex in $V_L(G)$ with

$R(u)$ colored S_i. Let Γ be the set of induced colorings on $B_{i,u}$. Note that Γ is the set of fixed H_{S_i}-colorings of $G_i = f_i(G, \epsilon/2)$. Also, each coloring in Γ comes up ψ times in $Y_{u,i}$ for some ψ. (In particular, ψ is the number of colorings of vertices other than $B_{i,u}$ which are induced by colorings in $Y_{u,i}$.) For coloring $y \in Y_{u,i}$ we will let $g(G, \epsilon, y) = g_i(G_i, \epsilon/2, y')$, where y' is the induced coloring on $B_{i,u}$. Then for every independent set x in the set $\mathcal{I}(G)$ of independent sets of G,

$$|\{y \in Y_{u,i} \mid g(G, \epsilon, y) = x\}| = \psi\, |\{y' \in \Gamma \mid g_i(G_i, \epsilon/2, y') = x\}|. \tag{11}$$

Since (f_i, g_i) is an SP-reduction, (1) gives

$$e^{-\epsilon/2} \frac{|\Gamma|}{|\mathcal{I}(G)|} \le |\{y' \in \Gamma \mid g_i(G_i, \epsilon/2, y') = x\}| \le e^{\epsilon/2} \frac{|\Gamma|}{|\mathcal{I}(G)|}, \tag{12}$$

and (5) follows for $Y_{u,i}$ from (11) and (12) since $|Y_{u,i}| = \psi\, |\Gamma|$. Colorings in $Y_{k+1}, \ldots, Y_{k+j}$ are handled similarly except that we look at induced colorings of $A_{i,u}$ rather than $B_{i,u}$.

It remains to satisfy (5) for $i = k + j + 1$. Note that any coloring y in Y_{k+j+1} points out an independent set of G. A vertex $u \in V_L(G)$ is in the independent set if $R(u)$ is colored $V_R(H)$. A vertex $v \in V_R(G)$ is in the independent set if $L(v)$ is colored $V_L(H)$. We will define $g(G, \epsilon, y)$ to be this independent set. Let us focus attention on a given independent set containing w_L vertices in $V_L(G)$ and w_R vertices in $V_R(G)$. We will now calculate how many colorings in Y_{k+j+1} correspond to this independent set.

For any bipartite graph G' with vertex partition $(V_L(G'), V_R(G'))$, let $\phi_H(G')$ denote the number of fixed H-colorings of G'. Then the number of times that this independent set comes up as a coloring in Y_{k+j+1} is the product of the following two quantities:

$$\left(\nu(b, v_R) f_L^a \prod_{i=1}^{k+j} \phi_H(A_{i,u}) \phi_H(A'_{i,u}) f_L^{a_i+b_i} v_R^{a_i+b_i}\right)^{w_L + v_R - w_R},$$

$$\left(f_R^b \nu(a, v_L) \prod_{i=1}^{k+j} \phi_H(B_{i,u}) \phi_H(B'_{i,u}) v_L^{a_i+b_i} f_R^{a_i+b_i}\right)^{v_L - w_L + w_R}.$$

Now note that $\phi_H(A_{i,u}) = \phi_H(B_{i,u})$ and $\phi_H(A'_{i,u}) = \phi_H(B'_{i,u})$. So if we let

$$Z = \left(\prod_{i=1}^{k+j} \phi_H(A_{i,u}) \phi_H(A'_{i,u})\right)^{v_L + v_R} (f_L^a \nu(b, v_R) f_L^q v_R^q)^{v_R} \left(f_R^b \nu(a, v_L) v_L^q f_R^q\right)^{v_L},$$

the contribution of the independent set becomes

$$Z(\nu(b, v_R) f_L^a f_L^q v_R^q)^{w_L - w_R} \left(f_R^b \nu(a, v_L) v_L^q f_R^q\right)^{w_R - w_L},$$

which is

$$Z\left(\frac{\nu(b, v_R) v_L^a}{v_R^b \nu(a, v_L)}\right)^{w_L - w_R} \left(\left(\frac{v_R}{f_R}\right)^{b+q} \left(\frac{f_L}{v_L}\right)^{a+q}\right)^{w_L - w_R}.$$

To get (8) we will show that a and b can be chosen so that

$$e^{-\epsilon/(8n)} \leq \left(\frac{\nu(b, v_R)v_L^a}{v_R^b \nu(a, v_L)}\right) \leq e^{\epsilon/(8n)} \tag{13}$$

and

$$e^{-\epsilon/(8n)} \leq \left(\frac{v_R}{f_R}\right)^{b+q}\left(\frac{f_L}{v_L}\right)^{a+q} \leq e^{\epsilon/(8n)}. \tag{14}$$

This guarantees that the contribution of this independent set is in the range $[e^{-\epsilon/4}Z, e^{\epsilon/4}Z]$, and (8) follows for Y_{k+j+1}. To establish (13), use Lemma 3 to observe that

$$\left(\frac{\nu(b, v_R)v_L^a}{v_R^b \nu(a, v_L)}\right) \leq \frac{1}{1 - \exp(-a/(2v_L))}.$$

Since (9) gives $1 - \exp(-a/(2v_L)) \geq 1 - \epsilon/(16n) \geq e^{-\epsilon/(8n)}$, the right-hand inequality of (13) follows. The left-hand inequality is similar.

We will now show how to choose the values of a and b to satisfy (14). If $v_R/f_R = v_L/f_L$, then simply choose $a = b$ and make them large enough to satisfy (9) and (10). Suppose that $v_R/f_R < v_L/f_L$. Then use Lemma 4 with $c_1 = v_R/f_R$, $c_2 = v_L/f_L$, $\delta = \epsilon/(8n)$, and

$$a_0 = 2v_L \left\lceil q \ln(v_R/f_R) + \ln(16n/\epsilon)\right\rceil + 2v_R \left\lceil q \ln(v_L/f_L) + \ln(16n/\epsilon)\right\rceil.$$

The lemma gives values of a and b which are in $O((a_0 + q)/\delta)$, which is not too large. Thus, our reduction is sampling-preserving. Note that the reduction can be done in polynomial time—the calculation of a and b does not involve computing Z. The case where $v_L/f_L < v_R/f_R$ is similar. □

6. The proof of Theorem 2. We start with some definitions. First, for every graph H we will define a loop-free bipartite graph $B[H]$ (this construction was used in [10]). Let the vertices of H be $v_1, \ldots, v_h$. The vertex set of $B[H]$ is $\{x_1, \ldots, x_h\} \cup \{y_1, \ldots, y_h\}$. The edge set of $B[H]$ is

$$\{(x_i, y_j) \mid (v_i, v_j) \in E(H)\}.$$

Thus, a loop (v_i, v_i) in H corresponds to the edge (x_i, y_i) in $B[H]$, and a nonloop (v_i, v_j) in H (for which $i \neq j$) corresponds to two edges (x_i, y_j) and (y_i, x_j) in $B[H]$. For every edge (v_i, v_j) of H, let

$$V_L(H_{i,j}) = \{x_\ell \mid (v_\ell, v_j) \in E(H)\}$$

and

$$V_R(H_{i,j}) = \{y_\ell \mid (v_i, v_\ell) \in E(H)\},$$

and let $H_{i,j}$ be the subgraph of $B[H]$ induced by vertex set $V_L(H_{i,j}) \cup V_R(H_{i,j})$. Note that $x_i \in V_L(H_{i,j})$ and $y_j \in V_R(H_{i,j})$ and x_i is adjacent to all of $V_R(H_{i,j})$ in $H_{i,j}$ and y_j is adjacent to all of $V_L(H_{i,j})$. Thus, $H_{i,j}$ is connected and full. Let $\Delta_1(H)$ be the degree of H. That is,

$$\Delta_1(H) = \max\{\deg(v) \mid v \in V(H)\}.$$

Similarly, let $\Delta_2(H)$ be the maximum degree amongst neighbors of vertices with degree $\Delta_1(H)$:

$$\Delta_2(H) = \max\{\deg(v) \mid \text{ for some } u \in V(H) \text{ with } \deg(u) = \Delta_1(H),\ (u,v) \in E(H)\}.$$

Let

$$R(H) = \{(v_i, v_j) \mid ((v_i, v_j) \in E(H) \text{ and } \deg(v_i) = \Delta_1(H) \text{ and } \deg(v_j) = \Delta_2(H)\}.$$

We will start with the following lemma.

LEMMA 6. *Let H be any fixed graph with no trivial components. Then $R(H)$ is nonempty, and $\Delta_1(H) > 1$ and $\Delta_2(H) > 1$. Also, for all $(v_i, v_j) \in R(H)$, $H_{i,j}$ is connected, loop-free, bipartite, full, and nontrivial.*

Proof. Since H has no trivial components, $R(H)$ is nonempty, and $\Delta_1(H) > 1$ and $\Delta_2(H) > 1$. Suppose $(v_i, v_j) \in R(H)$. Recall that $H_{i,j}$ is connected, loop-free, bipartite, and full. Suppose for contradiction that $H_{i,j}$ is a complete bipartite graph (so vertices in $V_L(H_{i,j})$ have degree $\Delta_1(H)$ in $H_{i,j}$ and vertices in $V_R(H_{i,j})$ have degree $\Delta_2(H)$ in $H_{i,j}$).

This assumption guarantees that $H_{i,j}$ is a connected component of $B[H]$: $B[H]$ cannot have an edge with exactly one endpoint in $V_L(H_{i,j})$—the endpoint would then have degree exceeding $\Delta_1(H)$ in $B[H]$, which is a contradiction; similarly, $B[H]$ cannot have an edge with exactly one endpoint in $V_R(H_{i,j})$.

Thus, for any $x_\ell \in V_L(H_{i,j})$,

$$\{v_r \mid (v_\ell, v_r) \in E(H)\} = \{v_r \mid y_r \in V_R(H_{i,j})\}. \tag{15}$$

Similarly, for any $y_\ell \in V_R(H_{i,j})$,

$$\{v_r \mid (v_\ell, v_r) \in E(H)\} = \{v_r \mid x_r \in V_L(H_{i,j})\}. \tag{16}$$

Now if H has a vertex v_ℓ such that $(v_i, v_\ell) \in E(H)$ and $(v_j, v_\ell) \in E(H)$, then $x_\ell \in V_L(H_{i,j})$ and $y_\ell \in V_R(H_{i,j})$, so (15) and (16) imply that

$$\{v_r \mid y_r \in V_R(H_{i,j})\} = \{v_r \mid x_r \in V_L(H_{i,j})\}.$$

Thus, $H_{i,j}$ corresponds to a component of H, and that component is a looped clique, which contradicts the fact that H has no trivial component.

On the other hand, if there is no v_ℓ with $(v_i, v_\ell) \in E(H)$ and $(v_j, v_\ell) \in E(H)$, then $H_{i,j}$ corresponds to a connected component of H which is a complete bipartite graph, again giving a contradiction. ∎

We can now prove the main lemma.

LEMMA 7. *Suppose that H is a fixed graph with no trivial components. Then* SAMPLEBIS $\leq_{\text{SP}}$ SAMPLEBH-COL.

Proof. Let (G, ϵ) be an input to SAMPLEBIS. For each $(v_i, v_j) \in R(H)$, Lemma 6 and Lemma 5 guarantee that there is a sampling-preserving reduction $(f_{i,j}, g_{i,j})$ from SAMPLEBIS to SAMPLEFIXED$H_{i,j}$-COL. Let $G_{i,j} = f_{i,j}(G, \epsilon/2)$. Let $f(G, \epsilon)$ be the graph which is constructed as follows. See Figure 3. Let $q = \sum_{(v_i,v_j)\in R(H)} |V_L(G_{i,j})| + |V_R(G_{i,j})|$. Let

$$V_L(f(G,\epsilon)) = A \cup \{w_L\} \cup \bigcup_{(v_i,v_j)\in R(H)} V_L(G_{i,j})$$

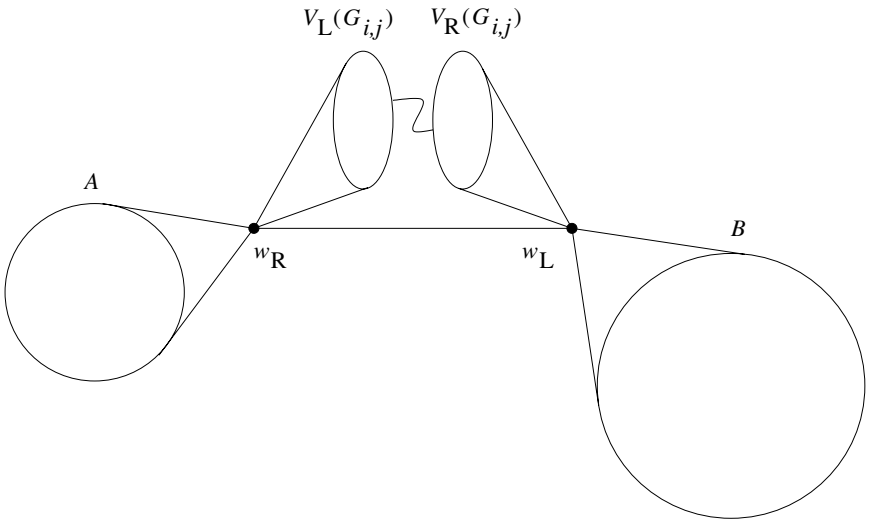


FIG. 3. *The construction of $f(G,\epsilon)$ in the proof of Lemma 7.*

and

$$V_R(f(G,\epsilon)) = B \cup \{w_R\} \cup \bigcup_{(v_i,v_j)\in R(H)} V_R(G_{i,j}),$$

where A and B are sets of vertices with

$$|A| = \left\lceil \frac{q\ln(|V(H)|) + \ln(8|E(H)|/\epsilon)}{\ln(\Delta_2(H)/(\Delta_2(H)-1))} \right\rceil$$

and

$$|B| = \left\lceil \frac{(q+|A|+1)\ln(|V(H)|) + \ln(8|V(H)|/\epsilon)}{\ln(\Delta_1(H)/(\Delta_1(H)-1))} \right\rceil.$$

Note that there is no division by zero, since $\Delta_1(H)$ and $\Delta_2(H)$ are bigger than one (by Lemma 6). In addition to the edges in the graphs $G_{i,j}$, we add edge (w_L, w_R) and $w_L \times B$ and $w_R \times A$ and, for all $(v_i, v_j) \in R(H)$, we add edges $w_L \times V_R(G_{i,j})$ and $w_R \times V_L(G_{i,j})$.

Let Y be the set of H-colorings of $f(G,\epsilon)$. Y_0 will be the set of colorings in Y in which (w_L, w_R) is not colored with an edge (v_i, v_j) from $R(H)$. We will now establish (7). For every $v \in V(H)$ with $\deg(v) < \Delta_1(H)$ let $Y_0(v)$ be the set of colorings in Y in which w_L is colored v. Now

$$|Y_0(v)| \le (\Delta_1(H)-1)^{|B|}|V(H)|^{q+|A|+1}.$$

Now consider any $(v_i, v_j) \in R(H)$. There are at least $\Delta_2(H)^{|A|}\Delta_1(H)^{|B|}$ colorings of $f(G,\epsilon)$ with (w_L, w_R) colored (v_i, v_j) (for example, the colorings in which all of the vertices of the graphs $G_{i,j}$ are colored with either v_i or v_j). Thus, $|Y| \ge \Delta_2(H)^{|A|}\Delta_1(H)^{|B|} \ge \Delta_1(H)^{|B|}$. We conclude that

$$|Y_0(v)| \le (\epsilon/(8|V(H)|))|Y|. \tag{17}$$

Now consider any edge $(v_i, v_k) \in E(H)$ such that $\deg(v_i) = \Delta_1(H)$ but $\deg(v_k) < \Delta_2(H)$. Let $Y_0(v_i, v_k)$ be the set of colorings in Y in which (w_L, w_R) is colored (v_i, v_k). Now

$$|Y_0(v_i, v_k)| \leq \Delta_1(H)^{|B|}(\Delta_2(H) - 1)^{|A|}|V(H)|^q.$$

Also, from before $|Y| \geq \Delta_2(H)^{|A|}\Delta_1(H)^{|B|}$, so

$$|Y_0(v_i, v_k)| \leq (\epsilon/(8|E(H)|))|Y|. \tag{18}$$

Equations (17) and (18) imply (7) since $|Y_0| \leq \sum_{v \in V(H)} Y_0(v) + \sum_{(v_i,v_k)} |Y_0(v_i, v_k)|$.

For an edge $(v_i, v_j) \in R(H)$, let $Y_{i,j}$ be the set of colorings of $f(G, \epsilon)$ with (w_L, w_R) colored (v_i, v_j). Let Γ be the set of induced colorings on $G_{i,j}$. Note that Γ is the set of fixed $H_{i,j}$-colorings of $G_{i,j}$. Also, each coloring in Γ comes up ψ times in $Y_{i,j}$, where ψ is the number of induced colorings on the vertices other than $G_{i,j}$. For a coloring $y \in Y_{i,j}$ we will set $g(G, \epsilon, y) = g_{i,j}(G_{i,j}, \epsilon/2, y')$, where y' is the induced coloring on $G_{i,j}$. Then (5) follows from the fact that $(f_{i,j}, g_{i,j})$ is an SP-reduction. □

Theorem 2 follows from Lemma 1 and Lemma 7 and from Lemma 8 below. Recall the following definitions. A *randomized approximation scheme* (RAS) for a counting problem F is a randomized algorithm that takes input σ and accuracy parameter $\epsilon \in (0, 1)$ and produces as output an integer random variable Y satisfying the condition $\Pr(e^{-\epsilon}F(\sigma) \leq Y \leq e^{\epsilon}F(\sigma)) \geq 3/4$. It is a "fully polynomial" randomized approximation scheme (FPRAS) if it runs in time $\text{poly}(|\sigma|, \epsilon^{-1})$. The problem #BIS is "self-reducible," so the following lemma follows from [17].

LEMMA 8 (JVV). *If* SAMPLEBIS *has a PAUS, then* #BIS *has an FPRAS.*

Proof. The lemma is a special case of Theorem 6.4 of [17]. In order to apply Theorem 6.4 directly we would need to define "self-reducible" formally and to deal with some easy (though annoying) issues:

(i) Inputs to #BIS may be disconnected, but inputs to SAMPLEBIS may not.
(ii) In order to apply Theorem 6.4 we technically need a *fully* polynomial almost uniform sampler (FPAUS) for SAMPLEBIS. This can be obtained from a PAUS, as [17] explains.

Rather than dealing with these issues, we prefer to simply provide a proof for the lemma. The details given here are from the proof of Proposition 3.4 of [16]. Technically, Jerrum's proof from [16] is about counting *matchings*, but the few changes that are needed to yield our lemma are completely routine.

Let (G, ϵ) be an input to #BIS. Suppose that G has components $G_1, \ldots, G_k$. For each i, let the two parts of the vertex set be $V_L(G_i)$ and $V_R(G_i)$, and let the sizes of these parts be ℓ_i and r_i, respectively. Let $N_i = \ell_i r_i$, and let $E(G_i) = \{e_i(1), \ldots, e_i(m_i)\}$. Denote the nonedges of G_i by $\{e_i(m_i + 1), \ldots, e_i(N_i)\}$. For $j \in \{1, \ldots, N_i\}$, let $G_i(j)$ be the graph $(V(G_i), \{e_i(1), \ldots, e_i(j)\})$. For any graph G', let $\mathcal{I}(G')$ denote the set of independent sets of G'. Let

$$\rho_i(j) = \frac{|\mathcal{I}(G_i(j+1))|}{|\mathcal{I}(G_i(j))|}.$$

Note that

$$|\mathcal{I}(G_i)| = (\rho_i(m_i)\rho_i(m_i + 1) \cdots \rho_i(N_i - 1))^{-1}|\mathcal{I}(G_i(N_i))|.$$

Also, the number of independent sets of the complete bipartite graph $G_i(N_i)$ is $2^{\ell_i} + 2^{r_i} - 1$, so

$$|\mathcal{I}(G_i)| = (2^{\ell_i} + 2^{r_i} - 1) \prod_{j=m_i}^{N_i-1} \rho_i(j)^{-1}. \tag{19}$$

Furthermore,

$$|\mathcal{I}(G)| = \prod_{i=1}^{k} |\mathcal{I}(G_i)| = \prod_{i=1}^{k} (2^{\ell_i} + 2^{r_i} - 1) \prod_{j=m_i}^{N_i-1} \rho_i(j)^{-1}. \tag{20}$$

Now let $z = \sum_{i=1}^{k} (N_i - m_i)$. In order to estimate $|\mathcal{I}(G)|$, we need to estimate the z ratios $\rho_i(j)$.

For each ratio $\rho_i(j)$ we can make some observations.

(i) $\rho_i(j) \leq 1$, since $\mathcal{I}(G_i(j+1)) \subseteq \mathcal{I}(G_i(j))$.

(ii) $\rho_i(j) \geq 1/2$, since $\mathcal{I}(G_i(j)) \backslash \mathcal{I}(G_i(j+1))$ can be mapped injectively into $\mathcal{I}(G_i(j+1))$ by removing the lexicographically least endpoint of $e_i(j+1)$.

(iii) Let $\mathcal{A}$ be a PAUS for SampleBIS. For $i \in [1, \ldots, k]$ and $j \in [m_i, \ldots, N_i - 1]$, let $Z_i(j)$ be the indicator variable for the event that, when we run $\mathcal{A}$ with input $G_i(j)$ and accuracy parameter δ, the output is an independent set of $G_i(j+1)$. Note that $\rho_i(j) - \delta \leq E[Z_i(j)] \leq \rho_i(j) + \delta$. This follows immediately from the definition of PAUS, but it is important to note that the input to $\mathcal{A}$, $G_i(j)$, is connected (since all inputs to SampleBIS must be connected).

Let $\overline{Z_i(j)}$ be the result obtained by calling $\mathcal{A}$ $\lceil 74\epsilon^{-2}z \rceil$ times with input $G_i(j)$ and accuracy parameter $\delta = \epsilon/(6z)$ and averaging the value of $Z_i(j)$ which occurs each time. Jerrum shows in his proof that with probability at least 3/4,

$$e^{-\epsilon} \prod_{i=1}^{k} \prod_{j=m_i}^{N_i-1} \rho_i(j) \leq \prod_{i=1}^{k} \prod_{j=m_i}^{N_i-1} \overline{Z_i(j)} \leq e^{\epsilon} \prod_{i=1}^{k} \prod_{j=m_i}^{N_i-1} \rho_i(j).$$

Thus, the quantity

$$\prod_{i=1}^{k} (2^{\ell_i} + 2^{r_i} - 1) \prod_{j=m_i}^{N_i-1} \overline{Z_i(j)}^{-1}$$

is a sufficiently accurate estimate of $|\mathcal{I}(G)|$.

For each of the z pairs (i, j), $O(\epsilon^{-2}z)$ samples were needed, each of which is produced in time $\text{poly}(|G|, z/\epsilon)$. Since $z \leq |V(G)|^2$, the total running time is $\text{poly}(|G|, \epsilon^{-1})$, and we have an FPRAS. □

7. The presence of trivial components. Theorem 2 shows that sampling H-colorings is difficult if every component of H is nontrivial. Recall from [10] that exactly counting H-colorings is #P-complete if H has even one nontrivial component. Thus, it might appear that Theorem 2 can be improved. In this section, we show that the existence of a single nontrivial component is not enough to make sampling difficult. In particular, we give an example of a graph H with a nontrivial component for which Sample$H$-Col has a PAUS. Specifically, let H be the graph depicted in Figure 4.

Observation 9. Suppose that H is the graph depicted in Figure 4. Sample$H$-Col has a PAUS.

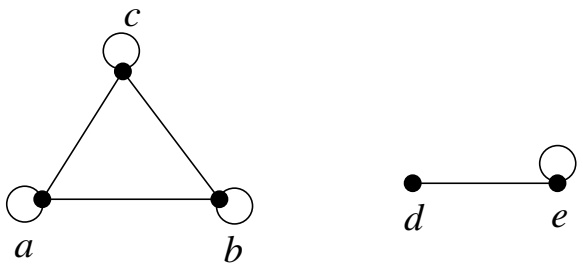


FIG. 4. *An H with a nontrivial component for which* SAMPLEH-COL *has a PAUS.*

Proof. Here is a PAUS for SAMPLEH-COL. The input is an instance (G, ϵ) where G has n vertices and, without loss of generality,[3] is connected. If $\epsilon < 2^n/(2^n + 3^n)$ then the algorithm simply runs for 5^n steps, constructs all of the H-colorings of G (and counts them) and selects one uniformly at random. Note that the running time is at most poly$(1/\epsilon)$ in this case. Otherwise, the algorithm chooses i uniformly at random from $1, \ldots, 3^n + 2^n$. If $i \leq 3^n$, then the algorithm outputs the i'th coloring from the 3^n colorings with colors "a", "b", and "c". Otherwise, let C be the $(i - 3^n)$th of the 2^n (proper and improper) colorings with colors "d" and "e". If C is a legal H-coloring of G, then the algorithm outputs it. Otherwise, it outputs the error symbol $\perp$. The variation distance between the output distribution of the algorithm and the uniform distribution on H-colorings of G is at most the probability that the algorithm outputs $\perp$, which is at most $2^n/(2^n + 3^n) \leq \epsilon$. □

8. Sampling and counting. Let #BH-COL be defined as follows.

Name. #BH-COL.
Instance. A loop-free connected bipartite graph G.
Output. The number of H-colorings of G.

For certain graphs H, the problem #BH-COL can be expressed as the counting problem associated with a "self-reducible p-relation." For such an H, Theorem 6.3 of Jerrum, Valiant, and Vazirani's paper [17] guarantees that if there is an FPRAS for #BH-COL, then there is a PAUS for SAMPLEBH-COL. If H has no trivial components, this in turn guarantees (by Theorem 2) an FPRAS for #BIS. Dyer and Greenhill [8] have given a more general framework in which these ideas work: If, for a given graph H, the problem #BH-COL is "self-partitionable," then an FPRAS for #BH-COL can be turned into a PAUS for SAMPLEBH-COL. It is not clear for which graphs H these ideas can be applied, and this is an interesting open question.

A related problem (which is also open) is to determine for which graphs H an FPRAS for counting H-colorings can be turned into a PAUS for SAMPLEH-COL. Dyer, Goldberg, and Jerrum [7] have shown that for every fixed H a PAUS for SAMPLEH-COL can be turned into an FPRAS for counting H-colorings.

Acknowledgments. We thank Martin Dyer, Paul Goldberg, and Mark Jerrum for useful conversations.

[3] We can assume that the input is a connected graph without losing generality because we can obtain an H-coloring of a k-component graph G by independently calling our PAUS for each component, specifying accuracy parameter ϵ/k. The final variation distance (between the output distribution and the uniform distribution on H-colorings of G) is at most ϵ.

REFERENCES

[1] C. Borgs, J. T. Chayes, M. Dyer, and P. Tetali, *On the sampling problem for H-colorings on the hypercubic lattice*, in Proceedings of the DIMACS-DIMATIA Workshop on Graphs, Homomorphisms and Statistical Physics, J. Nesetril and P. Winkler, eds., AMS, Providence, RI, to appear.

[2] G. R. Brightwell and P. Winkler, *Graph homomorphisms and phase transitions*, J. Combin. Theory Ser. B, 77 (1999), pp. 221–262.

[3] C. Cooper, M. Dyer, and A. Frieze, *On Markov chains for randomly H-coloring a graph*, J. Algorithms, 39 (2001), pp. 117–134.

[4] J. Diaz, M. Serna, and D. M. Thilikos, *(H, C, K)-coloring: Fast, easy, and hard cases*, in Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001), Marianske Lazne, Czech Republic, 2001, Lecture Notes in Comput. Sci. 2136, J. Sgall, A. Pultr, and P. Kolman, eds., Springer-Verlag, Heidelberg, 2001, pp. 304–315.

[5] J. Diaz, M. Serna, and D. M. Thilikos, *The complexity of parameterized H-colorings*, in Proceedings of the DIMACS-DIMATIA Workshop on Graphs, Homomorphisms and Statistical Physics, J. Nesetril and P. Winkler, eds., AMS, Providence, RI, to appear.

[6] M. Dyer, A. Frieze, and M. Jerrum, *On counting independent sets in sparse graphs*, SIAM J. Comput., 31 (2002), pp. 1527–1541.

[7] M. Dyer, L. A. Goldberg, and M. Jerrum, *Counting and sampling H-colorings*, Inform. Comput., to appear.

[8] M. Dyer and C. Greenhill, *Random walks on combinatorial objects*, in Surveys in Combinatorics, London Math. Soc. Lecture Note Ser. 267, J. D. Lamb and D. A. Preece, eds., Cambridge University Press, Cambridge, UK, 1999, pp. 101–136.

[9] M. Dyer and C. Greenhill, *On Markov chains for independent sets*, J. Algorithms, 35 (2000), pp. 17–49.

[10] M. Dyer and C. Greenhill, *The complexity of counting graph homomorphisms*, Random Structures Algorithms, 17 (2000), pp. 260–289.

[11] M. Dyer, C. Greenhill, L. A. Goldberg, and M. Jerrum, *On the relative complexity of approximate counting problems*, Algorithmica, to appear.

[12] M. Dyer, M. Jerrum, and E. Vigoda, *Rapidly mixing Markov chains for dismantleable constraint graphs*, in Proceedings of the DIMACS/DIMATIA Workshop on Graphs, Morphisms and Statistical Physics, J. Nesetril and P. Winkler, eds., AMS, Providence, RI, 2001, to appear.

[13] A. Galluccio, P. Hell, and J. Nešetřil, *The complexity of H-coloring of bounded degree graphs*, Discrete Math., 222 (2000), pp. 101–109.

[14] P. Hell and J. Nešetřil, *On the complexity of H-coloring*, J. Combin. Theory Ser. B, 48 (1990), pp. 92–110.

[15] M. Jerrum, *A very simple algorithm for estimating the number of k-colorings of a low-degree graph*, Random Structures Algorithms, 7 (1995), pp. 157–165.

[16] M. Jerrum, *Counting, Sampling and Integrating: Algorithms and Complexity*, Chapter 3, Birkhäuser-Verlag, Basel, 2003.

[17] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.

[18] E. Vigoda, *Improved bounds for sampling colorings*, J. Math. Phys., 41 (2000), pp. 1555–1569.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 433–447

THE POTENTIAL OF THE APPROXIMATION METHOD*

KAZUYUKI AMANO† AND AKIRA MARUOKA†

Abstract. Developing certain techniques for the approximation method, we establish precise versions of the following statements concerning lower bounds for circuits that detect cliques of size s in a graph with m vertices: For $5 \leq s \leq m/4$, a monotone circuit computing CLIQUE(m, s) contains at least $(1/2)1.8^{\min(\sqrt{s-1}/2, m/(4s))}$ gates: If a nonmonotone circuit computes CLIQUE using a "small" amount of negation, then the circuit contains an exponential number of gates. The former is proved by using so-called bottleneck counting argument within the framework of approximation, and the latter is verified by introducing a notion of restricting negation in circuits and generalizing these arguments to nonmonotone cases.

Key words. circuit complexity, lower bounds, clique function, approximation method, monotone circuit, negation-limited circuit

AMS subject classifications. 06E30, 68Q17, 68Q25, 94C10

DOI. 10.1137/S009753970138445X

1. Introduction.

Since Razborov introduced the approximation method [9] and used it successfully to prove a superpolynomial lower bound on the size of monotone circuits computing the clique function, much effort has been devoted to exploring the method and to deriving good lower bounds using it (e.g., [1, 2, 7, 8, 9, 10, 11, 13]). Alon and Boppana [1] obtained an exponential lower bound on the size of monotone circuits that compute the clique function, through a clever use of the approximation method. Using an apparently different argument, called *bottleneck counting*, Haken [4] derived an exponential lower bound on the size of monotone circuits that compute a variant of the clique function. Razborov, in subsequent work, showed that the approximation method can, in principle, provide tight lower bounds for nonmonotone circuits [10]. In spite of these advances, it remains a challenging problem to apply the method successfully to obtain good lower bounds on the size of circuits computing a given problem. This is especially hard when we deal with nonmonotone circuits.

In this paper, we extend our knowledge of the method in two ways. First, we show how to use the bottleneck counting argument within the framework of the approximation method. This allows us to present better lower bounds for a certain clique problem and to simplify considerably the proof both for the clique problem and for a problem that Alon and Boppana addressed. Second, we extend the method for nonmonotone circuits for the clique problem, although we must restrict ourselves to circuits with a restricted amount of negation.

More precisely, the results obtained in these ways are as follows. First, denoting by CLIQUE(m, s) the clique function detecting cliques of size s in a graph with m vertices, we obtain a lower bound of $(1/2)1.8^{\min(\sqrt{s-1}/2, m/(4s))}$ for $5 \leq s \leq m/4$. This lower bound should be contrasted with the best current lower bound for the clique function, $(1/8)(m/(4s^{3/2} \log m))^{\sqrt{s+1}/2}$ for $3 \leq s \leq (1/4)(m/\log m)^{2/3}$, obtained by

*Received by the editors February 6, 2001; accepted for publication (in revised form) October 24, 2003; published electronically February 24, 2004. An early extended abstract of this paper appeared in *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 431–440.

http://www.siam.org/journals/sicomp/33-2/38445.html

†Graduate School of Information Sciences, Tohoku University, Aoba 05, Aramaki, Aobaku, Sendai 980-8579, Japan (ama@ecei.tohoku.ac.jp, maruoka@ecei.tohoku.ac.jp).

Alon and Boppana [1]. So, as for the largest monotone lower bound for the clique function, our bound is $\exp(\Omega(m^{1/3}))$ for $s = \lceil m^{2/3} \rceil$, whereas the one due to Alon and Boppana is $\exp(\Omega((m/\log m)^{1/3}))$ for $s = (1/4)(m/\log m)^{2/3}$. To derive these lower bounds, we define approximators, instead of relying on the sunflower contraction, in terms of DNF and CNF formulas such that the size (or the length) of terms and clauses in the formulas is limited appropriately. In this way, we succeeded in simplifying greatly the proofs to obtain these bounds, using only elementary combinatorics. The similarity between the method of approximations and the bottleneck counting arguments were independently found by Berg and Ulfberg [3], Jukna [6], and Simon and Tsai [12]. More recently, Harnik and Raz [5] proved a higher lower bound of $2^{\Omega((n/\log n)^{1/3})}$ on the monotone circuit size of an explicit function by introducing the *monotone switching lemma* between DNF and CNF approximators.

Second, we extend the method discussed in the first part of this paper so as to apply for the case of nonmonotone circuits, provided that the amount of negation in circuits is restricted. To do so, we introduce a notion of restricting negation used in a circuit. For circuit C with input variables $x_1, \ldots x_n, \bar{x}_1, \ldots \bar{x}_n$, let its monotone analogue, denoted C_{ex}, be the monotone circuit obtained by replacing each negated variable $\bar{x}_i$ in C with a new input variable y_i. There exists an obvious correspondence between minterms of the function computed by a circuit C and those of the function computed by the corresponding circuit C_{ex}. When we deal with circuit C_{ex} we pay attention only to such minterms. Then we consider the maximum number of variables $y_1, \ldots, y_n$ appearing in such a minterm computed by C_{ex} as the parameter indicating an amount of negation used in C. Setting the parameter to an integer between 0 and n, we get a variety of restricted negation circuits, ranging from monotone circuits to general Boolean circuits. Based on the notion of restricting negation, we verify that if a nonmonotone circuit C computes CLIQUE(m, m^c) with the parameter at most $m^{c/2-\epsilon}$, then the size of the circuit C is given by $\exp(\Omega(m^{c/2}))$, where c and ϵ are any constants such that $0 < c \leq 2/3$ and $0 < \epsilon < c$. One might think of the result as showing the extent to which an argument based on bottleneck counting can be generalized to apply in nonmonotone cases.

2. Preliminaries. A *Boolean circuit* is a directed acyclic graph where each node has indegree 0 or 2. The nodes of indegree 0 are called input nodes, and are labeled with a variable x_i or its negation $\bar{x}_i$. The nodes of indegree 2 are called gates and are labeled with the Boolean functions AND and OR. AND and OR gates are also called $\wedge$ and $\vee$ gates, respectively. A circuit represents a Boolean function computed at a node in the circuit designated as the output node. In particular, a Boolean circuit without input nodes labeled with negated variables is called *monotone*. The *size* of a circuit C is the number of gates in the circuit C.

For ease of arguments, in section 3 a monotone circuit is further assumed to satisfy the following conditions: Any input of an $\vee$ gate is connected to either an output of $\wedge$ gate or an input node, and any input of an $\wedge$ gate is connected to either an output of $\vee$ gate or an input node. The output gate is an $\wedge$ gate. Any monotone circuit can be easily converted to the one satisfying these conditions by replacing, if necessary, a line connecting gates by a dummy gate (that can be thought of as an $\vee$ gate or an $\wedge$ gate, appropriately) with their two inputs being connected to an output of the same gate and hence by at most doubling the size of the circuit. The *circuit complexity* (*monotone circuit complexity*) of a function f is the size of the smallest circuit (monotone circuit) computing f.

3. Lower bounds for monotone circuit complexity based on CNF and DNF based approximators. In this section, we derive lower bounds for the clique problem and also those for a problem defined in terms of polynomials. The clique function, denoted CLIQUE(m, s), of $m(m-1)/2$ variables $\{x_{i,j} \mid 1 \le i < j \le m\}$ is defined to take value 1 if and only if the undirected graph on m vertices, with adjacency matrix X such that $x_{i,j}$ is the upper triangular submatrix of X, contains a clique of size s. The graphs will be identified with the assignments to the variables specifying the graphs.

THEOREM 3.1. *If $5 \le s \le m/4$, then any monotone circuit that computes the function CLIQUE(m, s) contains at least $(1/2)1.8^{\min(\sqrt{s-1}/2, m/(4s))}$ gates. In particular, the monotone circuit complexity of CLIQUE$(m, \lceil m^{2/3} \rceil)$ is $\exp(\Omega(m^{1/3}))$.*

We now proceed to proving Theorem 3.1 in the framework of approximation method. We define good and bad graphs which are used as test inputs to compare the circuit's behavior with the behavior of the clique function. A graph is called *good* if it consists of a clique on some set of s vertices and no other edges. A graph is called *bad* if there exists a partition of the vertices into $m \bmod (s-1)$ sets of size $\lceil m/(s-1) \rceil$ and $s - 1 - (m \bmod (s-1))$ sets of size $\lfloor m/(s-1) \rfloor$ such that any two vertices chosen from different sets have an edge between them, and no other edges exist. Note that the function CLIQUE(m, s) outputs value 1 on every good graph and outputs 0 on every bad graph. The following fact is obvious.

FACT 3.2. *There are*

$$\frac{m!}{s!(m-s)!}$$

good graphs and

$$\frac{m!}{(\lceil m/(s-1) \rceil !)^w (\lfloor m/(s-1) \rfloor !)^{s-1-w} w!(s-1-w)!}$$

bad graphs, where $w = m \bmod (s-1)$.

Let t be a term or a clause. The *endpoint set* of t is a set of all endpoints of the edges corresponding to variables in t. The *size* of t is the cardinality of the endpoint set of t.

We are ready to define an approximator circuit to approximate a monotone circuit. An approximator circuit $\overline{C}$ for a Boolean circuit C is the same graph as C, with $\vee$ and $\wedge$ gates replaced by $\overline{\vee}$ and $\overline{\wedge}$ gates. The nodes of the approximator circuit compute *approximators* defined as follows: the approximator corresponding to input variable x_i is defined to be x_i itself. The functions of an $\overline{\vee}$ gate and an $\overline{\wedge}$ gate are defined as follows (the integers l and r in the definition are chosen later):

- $\overline{\vee}$: Let f_1^D and f_2^D be two approximators, represented by monotone DNF formulas, feeding into an $\overline{\vee}$ gate. The approximate OR of these approximators is the monotone CNF formula obtained by transforming the monotone DNF formula $f_1^D \vee f_2^D$ into the equivalent monotone CNF formula and then taking away all the clauses whose sizes exceed r.
- $\overline{\wedge}$: Let f_1^C and f_2^C be two approximators, represented by monotone CNF formulas, feeding into an $\overline{\wedge}$ gate. The approximate AND of these approximators is the monotone DNF formula obtained by transforming the monotone CNF formula $f_1^C \wedge f_2^C$ into the equivalent monotone DNF formula and then taking away all the terms whose sizes exceed l.

Since we assumed that no output of an $\vee$ (resp., $\wedge$) gate is connected to an $\vee$ (resp., $\wedge$) gate, an approximator computed at an $\overline{\vee}$ gate is given by a monotone CNF

formula, and an approximator computed at an $\overline{\wedge}$ gate is given by a monotone DNF formula, where both formulas satisfy the size requirements. For Boolean functions f and g, let us denote $f \leq g$ if and only if $f(x) \leq g(x)$ holds for any input vector x. Note that the definition $(f_1 \,\overline{\vee}\, f_2) \geq (f_1 \vee f_2)$ holds for any monotone DNF formulas f_1 and f_2. Similarly, $(f_1 \,\overline{\wedge}\, f_2) \leq (f_1 \wedge f_2)$ holds for any monotone CNF formulas f_1 and f_2.

Let C be a monotone circuit computing CLIQUE(m, s), and let $\overline{C}$ (called the approximator circuit corresponding to C) denote the circuit obtained by replacing all of the $\vee$ and $\wedge$ gates in C by $\overline{\vee}$ and $\overline{\wedge}$ gates, respectively. Since the computation in $\overline{C}$ proceeds from bottom to top, it is easy to see that for any good graph that makes approximator circuit $\overline{C}$ output 0, there exists an $\overline{\wedge}$ gate that outputs 0 for the good graph due to taking the terms away in defining its output approximator. So, an $\wedge$ gate feeds the same input approximators as those to the $\overline{\wedge}$ gate outputs 1. Similarly, for any bad graph that makes approximator circuit $\overline{C}$ output 1, there exists an $\overline{\vee}$ gate that outputs 1 for the bad graph because of taking the clauses away in defining its output approximator.

The proof of Theorem 3.1 proceeds as follows: First, show that either of the number of good graphs that are incorrectly classified by the approximator circuit or the number of bad graphs incorrectly classified is large (Lemma 3.3). Second, show that the number of bad graphs for which an approximate gate $\vee$ (i.e., $\overline{\vee}$ gate) behaves differently from an $\vee$ gate is small (Lemma 3.4) and similarly the number of good graphs for which an approximate $\wedge$ gate (i.e., $\overline{\wedge}$ gate) behaves differently from a $\wedge$ gate is small (Lemma 3.5). Finally, calculate the numbers obtained by dividing the numbers of bad and good graphs incorrectly classified by an approximator circuit by the numbers of graphs for which the corresponding approximate gates behaves differently, respectively, and show that the larger of the two numbers calculated becomes large, completing the proof of Theorem 3.1. It is worthwhile to note that it is straightforward to extend our proofs to arbitrary fan-in circuit without changing the lower bounds we obtained.

The parameters l and r are chosen to be $l = \lfloor \sqrt{s-1}/2 \rfloor$ and $r = \lfloor m/(4s) \rfloor$.

LEMMA 3.3. *An approximator circuit either outputs identically* 0 *or outputs* 1 *on at least one half of the bad graphs.*

Proof. Let $\overline{f}$ be the approximator function that an approximator circuit computes. Because of the assumption that the output gate of an approximator circuit is $\wedge$ gate, $\overline{f}$ can be represented by a monotone DNF formula consisting of terms of size at most l. If $\overline{f}$ is identically 0, then the first conclusion holds. If not, then there is a term t whose size is less than or equal to l such that $\overline{f} \geq t$ holds. In what follows, bad graphs are represented as one-to-one mappings from vertex set $\{v_1, \ldots, v_m\}$ to $\{(1,1),\ldots,(1, \lceil m/(s-1) \rceil), \ldots, (w,1),\ldots,(w, \lceil m/(s-1) \rceil), (w+1,1), \ldots, (w+1, \lfloor m/(s-1) \rfloor), (w+2,1), \ldots, (s-1, \lfloor m/(s-1) \rfloor)\}$. Such a mapping specifies a bad graph in the obvious way: Two vertices in the graph have an edge between them if and only if the mapping assigns to the vertices a pair with different first components. Note that there exist many mappings corresponding to one bad graph. It is easy to see that the ratio of mappings that satisfy the condition that there is a variable x in the term t such that two vertices incident to x are assigned a pair with the same first component, i.e., the term t outputs 0 on bad graphs specified by such mappings, is at most $(l(l-1)/2)(\lceil m/(s-1) \rceil / m)$. Recalling $l = \lfloor \sqrt{s-1}/2 \rfloor$, the quantity above is bounded from above by $1/2$. Therefore, the ratio of bad graphs such that $\overline{f}$ outputs 1 on them is at least $1/2$. □

LEMMA 3.4. *Suppose that an $\vee$ gate and an $\overline{\vee}$ gate are given as input the same monotone DNF formulas such that the size of terms in the formulas is equal to or less than l. Then the number of bad graphs for which the $\vee$ and $\overline{\vee}$ gates produce different outputs (the $\vee$ gate produces 0, whereas the $\overline{\vee}$ gate produces 1) is at most*

$$\frac{(m/2)^{r+1}(m-r-1)!}{(\lceil m/(s-1)\rceil !)^w(\lfloor m/(s-1)\rfloor !)^{s-1-w}w!(s-1-w)!}. \tag{3.1}$$

Proof. Suppose that an $\vee$ gate and an $\overline{\vee}$ gate are given as input the same monotone DNF formulas, denoted f_1^D and f_2^D, such that the size of any term in the formulas is equal to or smaller than l. Let $f_1^D \vee f_2^D$ and $f_1^D \,\overline{\vee}\, f_2^D$ be denoted by f^D and f^C, respectively. Let $t_1, \ldots, t_q$ be the complete list of terms in f^D. Then each t_i contains at most $l(l-1)/2$ variables. We shall count the number of bad graphs x such that both $f^D(x) = 0$ and $f^C(x) = 1$ hold. As in the proof of Lemma 3.3, bad graphs are represented as mappings described there. Instead of counting bad graphs directly, we count mappings corresponding to bad graphs. The number in question is bounded from above by the number of the mappings corresponding to bad graphs x such that $f^D(x) = 0$ and $f^C(x) = 1$ divided by the number of mappings corresponding to one bad graph. (The number of mappings is independent of the bad graph chosen.) Since the latter is given by the denominator of (3.1), it suffices to estimate the former. The former is the number of mappings corresponding to bad graphs x that do not satisfy any of $t_1, \ldots, t_q$ but satisfy all clauses in f^C. We count how many ways one could choose variables from terms t_1 up to t_q and assign pairs of integers to the endpoints associated with the variables chosen so that the corresponding bad graphs x satisfy $f^D(x) = 0$ and $f^C(x) = 1$.

Suppose that we proceed to term t_i, and hence some of the endpoints associated with variables from terms t_1 to t_{i-1} are already assigned distinct pairs of integers. This partial assignment assigns 0 and 1 to some of variables in the way mentioned above. We first consider the extreme cases. If there exists a variable in term t_i already assigned 0 by the partial assignment, we skip to the next term t_{i+1}. The other extreme case occurs when all the variables in term t_i are assigned 1 so far. In this case term t_i will never take value 0, hence we don't need to consider the case.

If neither of these extreme cases happens, choose a variable from term t_i such that at least one of the vertices associated with the variable is not assigned a pair of integers. There are two cases to consider: If exactly one of the vertices is assigned so far, then assign to the remaining vertex a pair whose first component is identical to the first component of the pair of integers assigned to the other vertex so that the variable associated with the two vertices is assigned 0. In this case, there are at most $\lceil m/(s-1)\rceil - 1 \le m/(s-1)$ ways of assigning pairs of integers to the vertex. On the other hand, if both of the vertices have not been chosen so far, assign to these vertices pairs of integers with the same first components, so that the variable associated with the two vertices is assigned 0. So, for the two vertices, there are at most $(s-1)(\lceil m/(s-1)\rceil)(\lceil m/(s-1)\rceil - 1) \le 2m^2/(s-1)$ ways of assigning pairs of integers.

Suppose that there exist k variables in term t_i such that exactly one of the vertices corresponding to the variables is assigned a pair of integers so far. Then there exist at most $l(l-1)/2 - k$ variables in term t_i such that none of the vertices associated with the variables is assigned a pair of integers so far. So the number of ways of choosing an unassigned vertex in the endpoints of variables in t_i and assigning a pair of integers

to the chosen vertex is bounded from above by

$$\max_k(km/(s-1) + \sqrt{(l(l-1)/2-k)(2m^2/(s-1))}), \tag{3.2}$$

where integer k ranges from 0 to $l(l-1)/2$. This is because doing something to two vertices in i ways can be regarded as doing something to a vertex appropriately in $\sqrt{i}$ ways twice successively. Because of $l \leq \sqrt{s-1}/2$, a simple calculation shows that the quantity above is maximized when $k = 0$, and is bounded from above by $m/2$.

By the definition of $\overline{\vee}$ gate, it is easy to see that the graphs x corresponding to the mappings specified in this way satisfy $f^D(x) = 0$ and $f^C(x) = 1$ only if there exist more than r vertices assigned pairs of integers in the above procedure. So the number of mappings corresponding to such bad graphs is bounded from above by the number of mappings such that $r+1$ vertices are assigned pairs of integers multiplied by the number of ways of assigning arbitrarily distinct pairs of integers to the remaining $m-r-1$ vertices. The resulting number is given by the numerator of (3.1), completing the proof. □

LEMMA 3.5. *Suppose that an $\wedge$ gate and an $\overline{\wedge}$ gate are given as input the same monotone CNF formulas such that the size of terms in the formulas is equal to or less than r. Then the number of good graphs for which the $\wedge$ gate and the $\overline{\wedge}$ gate produce different outputs (the $\wedge$ gate produces 1, whereas the $\overline{\wedge}$ gate produces 0) is at most*

$$\frac{(m/2)^{l+1}(m-l-1)!}{s!(m-s)!}.$$

Proof. The proof is similar to that of Lemma 3.4. Suppose that an $\wedge$ gate and an $\overline{\wedge}$ gate are given as input the same monotone CNF formulas, denoted f_1^C and f_2^C, as in the lemma. Let $f^C = f_1^C \wedge f_2^C$ and $f^D = f_1^C \,\overline{\wedge}\, f_2^C$. Let $c_1, \ldots, c_q$ be the complete list of clauses in f^C, where each c_i contains at most $r(r-1)/2$ variables. The number in question is equal to the number of good graphs x such that $f^C(x) = 1$ and $f^D(x) = 0$ hold.

Instead of the mappings from vertices to the integer pairs in the case of Lemma 3.4, we consider one-to-one mappings from the vertex set $\{v_1, \ldots, v_m\}$ to the integer set $\{1, \ldots, m\}$. Such a mapping can be thought of as specifying a good graph such that the set of vertices assigned integers from 1 to s forms a clique. The number mentioned in the lemma is at most the number of mappings corresponding to good graphs x that satisfy all clauses of $c_1, \ldots, c_q$ but do not satisfy any term in f^D divided by the number of mappings corresponding to one good graph.

As in the case of Lemma 3.4, we count how many ways one could choose variables each from clauses of $c_1, \ldots, c_q$ and assign integers to the vertices associated with the variables chosen so that all of the clauses are satisfied but the disjunctive normal form formulas $f_1^C \,\overline{\wedge}\, f_2^C$ is not satisfied. Suppose that we proceed to clause c_i and, hence, some of the vertices are assigned integers from 1 to s so that all of the clauses from c_1 to c_{i-1} are satisfied. Such an assignment is called a partial assignment. If there exists a variable in c_i assigned 1 by the partial assignment, we skip to the next clause c_{i+1}.

The variable $x_{j,k}$ is said to be incident to vertices v_j and v_k. To count the number of ways of making clause c_i take value 1, there are two cases to consider: Choose a variable in c_i incident to two vertices; one is assigned an integer by the partial assignment and the other is not. Choose a variable in c_i incident to two vertices that are not assigned integers so far.

As in the proof of Lemma 3.4, the number of ways of choosing unassigned vertices and assigning integers to the vertices so as to make clause c_i take value 1 is bounded from above by

$$r(s-1)+\sqrt{(r(r-1)/2)s(s-1)} < 2rs. \tag{3.3}$$

Since $r \le m/(4s)$, this quantity is bounded from above by $m/2$. The rest of the proof is similar to that of Lemma 3.4. □

Now we proceed to the proof of Theorem 3.1.

Proof of Theorem 3.1. In view of Fact 3.2 and Lemmas 3.3, 3.4, and 3.5, the size of a monotone circuit that computes CLIQUE(m,s) is at least

$$\min\left(\frac{m!}{2(m/2)^{r+1}(m-r-1)!}\ ,\ \frac{m!}{(m/2)^{l+1}(m-l-1)!}\right),$$

which is bounded from below by

$$\min\left(\frac{(m-r)^{r+1}}{2(m/2)^{r+1}}\ ,\ \frac{(m-l)^{l+1}}{(m/2)^{l+1}}\ \right). \tag{3.4}$$

Since $5 \le s \le m/4$, we have $r \le m/10$ and $l \le m/10$. Hence, $m-r \ge 9m/10$ and $m-l \ge 9m/10$ hold. Therefore, (3.4) is bounded from below by

$$\min(1.8^{r+1}/2\ ,\ 1.8^{l+1}).$$

Thus, noticing $r+1 \ge m/(4s)$ and $l+1 \ge \sqrt{s-1}/2$, the proof is completed. □

One might expect that the definitions of approximate operations in the proof of Theorem 3.1 can be simplified by replacing "size" with "length," which is the number of variables appearing in terms and clauses. However, it turns out that such a simplification yields a weaker (although still exponential) lower bound.

Consider that we define the approximate operations based on "length" with suitable choices of l and r and apply the same argument as in the proof of Theorem 3.1. By the same counting argument as in the proof of Lemmas 3.4 and 3.5, the upper bound of the number of choices corresponding to (3.2) in the proof of Lemma 3.4 is

$$\max_k\{km(s-1)+\sqrt{(l-k)(2m^2/(s-1))}\}, \tag{3.5}$$

and the one corresponding to (3.3) in the proof of Lemma 3.5 is

$$\max_k\{k(s-1)+\sqrt{(r-k)s(s-1)}\}. \tag{3.6}$$

It is easy to see that if $r=\lfloor m/(4s)\rfloor$ and $l=\lfloor (s-1)/8\rfloor$, then the quantities of (3.5) and (3.6) are both smaller than $m/2$. It is also easy to see that the Lemma 3.3 holds for such l. However, Lemmas 3.4 and 3.5 may not hold this time. For example, in the proof of Lemma 3.4, the condition that there exist more than r vertices assigned pairs of integers in the procedure described in the proof is not a necessary condition to satisfy $f^D(x)=0$ and $f^C(x)=1$. Because if $r'(r'-1)/2 > r$, then there is a clause with more than r variables corresponding to the assignment that only r' vertices assigned pairs of integers. So we must replace the variable r in the statement of Lemma 3.4 with r' such that $r'(r'-1)/2 \le r$. By a similar reason, we also have to replace the variable l in the statement of Lemma 3.5 with l' such that $l'(l'-1)/2 \le l$. By using these lemmas and the similar argument as in the proof of Theorem 3.1,

we can obtain the $\exp(\Omega(\min(r', l')))) = \exp(\Omega(\min(\sqrt{m/s}, \sqrt{s})))$ lower bound for the size of a monotone circuit that computes CLIQUE(m, s). This lower bound is an exponential in m for a suitable choice of s; however, it is slightly weaker than the $\exp(\Omega(\min(m/s, \sqrt{s})))$ lower bound of Theorem 3.1.

We now consider another problem to show that our CNF and DNF based approximators make lower bound proofs simple.

Let $n = q^2$ and $x = \{x_{i,j} \mid 1 \leq i, j \leq q\}$, where q is a prime. Let $G(x)$ be the bipartite graph on $V = \{v_1, \ldots, v_q\}$ and $W = \{w_1, \ldots, w_q\}$ with edge set $\{(v_i, w_j) \mid x_{i,j} = 1\}$. The function POLY$(q, s)(x)$ of q^2 variables $\{x_{i,j}\}$ is defined as POLY$(q, s)(x) = 1$ if and only if there is a polynomial p over the field Z_q of degree at most $s - 1$ such that $G(x)$ includes all edges $(v_i, w_{p(i)})$ for $1 \leq i \leq q$.

The next theorem gives the same lower bound as the one due to Alon and Boppana [1].

THEOREM 3.6 (Alon and Boppana [1]). *If $s \leq (1/2)\sqrt{q/\ln q}$, then any monotone circuit that computes the function POLY(q, s) contains $q^{\Omega(s)}$ gates.*

To prove Theorem 3.6, we need a few lemmas. A graph is called *good* if there exists a polynomial p of degree at most $s - 1$ such that the set of edges of the graph is given as $\{(v_i, w_{p(i)}) \mid 1 \leq i \leq q\}$. Notice that the number of good graphs is q^s. For $0 \leq \epsilon \leq 1/2$, let NG$_\epsilon$ be the probability distribution on bipartite graphs with each edge appearing independently with probability $1 - \epsilon$. We choose $\epsilon = (2s \ln q)/q$. It is easy to see that

$$\Pr_{x \in NG_\epsilon}[\text{POLY}(q, s)(x) = 1] \leq q^s(1 - \epsilon)^q \leq q^s e^{-\epsilon q} = 1/q^s \leq 1/4. \tag{3.7}$$

So a graph x chosen according to NG$_\epsilon$ makes the value of the function POLY(q, s) to 0 with probability at least 3/4. These graphs serve as the bad graphs in the previous case.

Instead of defining approximators in terms of the size of terms and clauses, we define approximators this time in terms of the length of terms and clauses, i.e., the number of variables appearing in terms and clauses. Approximate operations of $\overline{\vee}$ gate and $\overline{\wedge}$ gate are defined exactly the same way as the previous case except that "size" is replaced with "length." As in the previous case, approximators are defined in bottom-up fashion, taking the approximator corresponding to input variable x_i to be x_i itself. The parameters l and r are put this time as $l = s$ and $r = \lceil \sqrt{q \ln q} \rceil$.

To prove Theorem 3.6, we need Lemmas 3.7, 3.8, and 3.9, which correspond to Lemmas 3.3, 3.4, and 3.5, respectively. We now proceed to the proof of the theorem.

LEMMA 3.7. *Let $\overline{f}$ be a function computed by an approximator circuit. Then $\overline{f}$ is identically* 0 *or*

$$\Pr_{x \in NG_\epsilon}[\textit{POLY}(q, s)(x) = 0 \textit{ and } \overline{f}(x) = 1] \geq 1/4.$$

Proof. If $\overline{f}$ is identically 0, then the first conclusion holds. If not, then there exists a term t of length at most l such that $\overline{f} \geq t$. Recall that the output gate is assumed to be an $\wedge$ gate, and, hence, $\overline{f}$ is given as a monotone DNF formula. Since $\epsilon = (2s \ln q)/q$, $l = s$ and $s \leq (1/2)\sqrt{q/\ln q}$, we have $\epsilon l \leq 1/2$. Since $0 \leq \epsilon 1/2$, we therefore have

$$\Pr_{x \in NG_\epsilon}[\overline{f}(x) = 1] \geq (1 - \epsilon)^l \geq (1/4)^{\epsilon l} \geq (1/4)^{1/2} = 1/2. \tag{3.8}$$

Thus the lemma follows from (3.8) and (3.7). □

LEMMA 3.8. *Let f_1^D and f_2^D be monotone DNF formulas such that the length of each term in them is at most l. Then*

$$\Pr_{x \in NG_\epsilon}[(f_1^D \vee f_2^D)(x) = 0 \text{ and } (f_1^D \overline{\vee} f_2^D)(x) = 1] \leq (1/2)^r.$$

Proof. Let f^D denote the monotone DNF formula $f_1^D \vee f_2^D$. We construct a decision tree computing f^D as follows. We start with a vertex associated with f^D, which becomes the root of the decision tree we shall construct. Take a term whose length is, say i, out of terms in f^D, and construct the path consisting of i vertices labeled with variables in the term and a leaf labeled 1. Furthermore, all the edges on the path is labeled with 1. Draw i edges out of all the vertices except the leaf together with distinct endpoints added and label these edges with 0. Each path from the root to one of the i vertices added specifies in the obvious way an assignment of 1 and 0 to the variables on the path except the endpoint of the path. Associate these endpoints with the functions obtained by substituting the Boolean values to the variables of f^D according to the corresponding assignments. Then repeat the procedure mentioned above with the vertices added until all the vertices are assigned with the constant functions.

Let $path_x$ denote the path from the root to a leaf specified, in the obvious way, by assignment x. Let (u, v) be an arbitrary edge labeled with 0 on $path_x$. Clearly, the probability that the $path_x$ doesn't pass any edge labeled with 0 after (u, v) on the condition that $path_x$ passes edge (u, v) is at least $(1 - \epsilon)^l$. This is because $path_x$ has at most l consecutive edges labeled with 1. Therefore, the probability that $path_x$ passes more that r edges labeled with 0 is at most $(1 - (1 - \epsilon)^l)^r$. Clearly, if both $(f_1^D \vee f_2^D)(x) = 0$ and $(f_1^D \overline{\vee} f_2^D)(x) = 1$ hold, then $path_x$ passes more than r edges labeled with 0. So the probability mentioned in the lemma is bounded from above by $(1 - (1 - \epsilon)^l)^r$. From the inequality $(1 - \epsilon)^l \geq 1/2$ in (3.8), this quantity is at most $(1 - 1/2)^r = (1/2)^r$. This completes the proof. □

LEMMA 3.9. *Let an $\wedge$ gate and an $\overline{\wedge}$ gate be given as input the same monotone CNF formulas such that the length of clauses in the formulas is equal to or less than r. Then the number of good graphs for which the $\wedge$ gate and the $\overline{\wedge}$ gate produce different outputs is at most r^l.*

Proof. The proof is similar to that of Lemma 3.5. Suppose that an $\wedge$ gate and an $\overline{\wedge}$ gate are given as input the same monotone CNF formulas, denoted f_1^C and f_2^C, such that the size of any clause in the formulas is equal to or less than r. Let $f_1^C \wedge f_2^C$ be denoted by f^C and let $c_1, \ldots, c_q$ be the complete list of clauses of f^C. Let f^D denote the DNF formula equivalent to f^C. Clearly, f^D is the Boolean sum of terms of distinct variables, each being chosen from c_1 up to c_p. Since $f_1^C \overline{\wedge} f_2^C$ is equal to the DNF formula obtained from f^D by taking away all the terms whose sizes exceed l and $l(= s)$ variables $x_{i,j}$ with distinct first indices i's specify a polynomial with degree $s - 1$, hence a good graph, it is easily seen that the number of good graphs x's such that $(f_1^C \wedge f_2^C)(x) = 1$ and $(f_1^C \overline{\wedge} f_2^C)(x) = 0$ is bounded from above by the number of ways of choosing a variable out of r variables (appearing in a clause) l times. Thus the number of such good graphs is at most r^l, completing the proof. □

Theorem 3.6 easily follows from Lemmas 3.7, 3.8, and 3.9.

Proof of Theorem 3.6. In view of Lemmas 3.7, 3.8, and 3.9, it is concluded that the size of the monotone circuit computing POLY(q, s) is at least $\min((q/r)^l, (1/4)2^r)$. Thus, the equations $l = s$ and $r = \lceil \sqrt{q \ln q} \rceil$ give the desired bound, completing the proof. □

4. Lower bounds for nonmonotone circuit complexity based on negation-based approximators. For two vectors v and v' in $\{0,1\}^n$, we denote $v' \lneq v$ if $v \neq v'$ and $v'_i \leq v_i$ for any $1 \leq i \leq n$, where v_i denotes the ith component of v. A vector v is called a *minimal true vector* for a Boolean function f if $f(v) = 1$ and, for any $v' \lneq v$, $f(v') = 0$. Let $\mathrm{Min}(f)$ be the set of all minimal true vectors of f. Similarly, when circuit C computes a function f, $\mathrm{Min}(C)$ denotes $\mathrm{Min}(f)$. Given a nonmonotone circuit C with input literals $x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n$, let C_{ex} denote the monotone circuit obtained from C by simply replacing each negated input variable $\bar{x}_i$ in C with a new input variable y_i for $1 \leq i \leq n$ without making any further modification. In what follows, C_{ex} will be called the monotone analogue of the original circuit C. Let the input vectors to circuits C and C_{ex} be denoted by $(x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n)$ and $(x_1, \ldots, x_n, y_1, \ldots, y_n)$, respectively. Obviously, these circuits produce the same output values for inputs represented as $(x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n)$.

Suppose that a circuit C computes a function f. When an input vector is given, all the wires in C are assigned 0 or 1 in the obvious way. We call a wire assigned the value 1 an *activated* wire. Given a positive vector, i.e., a vector that makes circuit C output 1, we have the collection of activated wires in the circuit C which contains at least one path from an input node to the output node. On the other hand, in the case of a negative vector, i.e., a vector that makes C output 0, such a collection of wires does not contain any path from an input node to the output node.

Now suppose that a positive vector x is given to the circuit C. In the nonmonotone case, not only the wires connected to inputs x_i with $x_i = 1$, but also the ones connected to inputs $\bar{x}_i$ with $x_i = 0$, which do not appear in monotone circuits, are activated. Since the circuit C contains only AND and OR gates in its inside, the entire collection of activated wires increases monotonically as the collection of activated wires at the input level increases. Hence, viewing the computation of a circuit like this, one might imagine that the more negated variables we are allowed to use, the more economically the collections of activated wires can reach the output node. These considerations lead us to the following definition, which formalizes a measure of the amount of negation used in the process of computation in a circuit.

DEFINITION 4.1. *Let C be a circuit computing a monotone function f on n variables and let C_{ex} be the monotone analogue of C. For v and u in $\{0,1\}^n$, $|v|_1$ denotes the number of 1's among v, and $v \circ u$ denotes the concatenation of v and u. For $0 \leq k \leq n$, we define $\mathrm{Min}_k(C)$ as follows:*

$$\mathrm{Min}_k(C) = \{v \in \mathrm{Min}(f) \,|\exists y \in \{0,1\}^n (|y|_1 \leq k) \wedge (v \circ y \in \mathrm{Min}(C_{ex}))\}.$$

In other words, a minterm v in f is in $\mathrm{Min}_k(C)$ when the minimum number of variables $y_1, \ldots, y_n$ appearing in the minterm of the function computed by C_{ex}, which corresponds to the minterm v, is at most k.

Note that by the definition

$$\mathrm{Min}_0(C) \subseteq \mathrm{Min}_1(C) \subseteq \cdots \subseteq \mathrm{Min}_n(C) = \mathrm{Min}(f).$$

In particular, if C is monotone, then $\mathrm{Min}_0(C) = \mathrm{Min}(f)$. We can think of the minimum i satisfying $\mathrm{Min}_i(C) = \mathrm{Min}(f)$ as a sort of measure indicating an amount of negation used in computing f. The condition $\mathrm{Min}_{m^{c/2-\epsilon}}(C) = \mathrm{Min}(\mathrm{CLIQUE}(m, m^c))$ in the theorem below can be thought of as saying that C uses a "small" amount of negation in the sense of Definition 4.1 because $m^{c/2-\epsilon}$ is much smaller than $n = m(m-1)/2$, namely the number of variables of CLIQUE. So, roughly, the theorem says that if a

nonmonotone circuit computes the clique function using a small amount of negation, then the circuit has an exponential number of gates.

THEOREM 4.2. *Let c and ϵ be constants such that $0 < c \leq 2/3$ and $0 < \epsilon < c/2$. Suppose that a circuit C computes the function CLIQUE(m, m^c). If $\mathrm{Min}_{m^{c/2-\epsilon}}(C) = \mathrm{Min}(CLIQUE(m, m^c))$ holds, then the size of C is $\exp(\Omega(m^{c/2}))$.*

The proof of the theorem will be done by a generalized argument based on the approximation method.

Let f be the function CLIQUE(m, s), where $s = m^c$. Let C be a circuit computing f whose monotone analogue is denoted by C_{ex}. Without loss of generality, we assume that m is divisible by s. Note that C_{ex} is assumed to be layered with alternating AND and OR layers, i.e., every AND gate is connected to an OR gate and vice versa, and the output gate of C_{ex} is an AND gate. Let $n = \binom{m}{2}$ be the number of input variables of CLIQUE. A vector $v \circ y$ of length $2n$ is called a *good* vector if $v \in \mathrm{Min}(f)$, $|y|_1 \leq k$, and $v \circ y \in \mathrm{Min}(C_{ex})$, where k is an integer whose value will be determined later. By the definition of $\mathrm{Min}(C_{ex})$, if $v \circ y$ is a good vector, then $v \circ y' \notin \mathrm{Min}(C_{ex})$ for any $y' \lneq y$. Let V be the set of m vertices of the graph associated with CLIQUE. A one-to-one mapping from V to the set $\{0, 1, \ldots, s-1\} \times \{1, \ldots, m/s\}$ is called a *bad* mapping. The definition of the bad mapping is the same to that for the monotone case except that we added the value 0 to the range of the first component of the bad mapping. We consider a bad mapping ϕ as the graph that has an edge between u and v whenever the first elements of $\phi(u)$ and $\phi(v)$ are different and both of them are not 0. For each bad mapping ϕ, the bad vector associated with ϕ is the vector $u \circ \bar{u}$, where $\bar{u} = (u\bar{}_1, u\bar{}_2, \ldots, u\bar{}_n)$. Note that every good vector makes the circuit C_{ex} output 1, whereas every bad vector makes the circuit C_{ex} output 0. The following fact is obvious.

FACT 4.3. *There are at least $|\mathrm{Min}_k(C_{ex})|$ good vectors and $m!$ bad mappings.*

We are now ready to define approximators as well as approximate operations. Put $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$. Let $\mathcal{N}(k)$ denote the collection of subsets of Y of size at most k. For a set of variables $L \subseteq X$, let $\lfloor L \rfloor$ denote the set of all endpoints of the edges corresponding to variables in L. For $W \subseteq V$, the set of variables whose both endpoints are in W is denoted by $\lceil W \rceil$. $\lceil W \rceil$ can be thought of as the subset of X which corresponds to the clique on W. As in the previous section, the *size* of a term or a clause t is the cardinality of the endpoint set of variables in t. In what follows, for a set of literals L, the product of all literals in L is denoted by $\wedge L$.

An approximator is defined to be a set of monotone functions denoted $\{f_Y\}_{Y \in \mathcal{N}(k)}$, where all of f_Y's are either monotone DNF formulas on X or monotone CNF formulas on X. We interpret the approximator $\{f_Y\}_{Y \in \mathcal{N}(k)}$ to represent the function written as $\vee_{Y \in \mathcal{N}(k)}(\wedge Y \cdot f_Y)$.

- $\overline{\vee}$: Let $\{f^D_{1,Y}\}_{Y \in \mathcal{N}(k)}$ and $\{f^D_{2,Y}\}_{Y \in \mathcal{N}(k)}$ be two approximators, each being sets of monotone DNF formulas. The approximate OR of these approximators, denoted $\{f^D_{1,Y}\}_{Y \in \mathcal{N}(k)} \overline{\vee} \{f^D_{2,Y}\}_{Y \in \mathcal{N}(k)}$, is defined as the set of monotone CNF formulas, denoted $\{f^C_Y\}_{Y \in \mathcal{N}(k)}$, where, for each $Y \in \mathcal{N}(k)$, f^C_Y is the monotone CNF formula obtained by transforming $f^D_{1,Y} \vee f^D_{2,Y}$ into the equivalent monotone CNF formula and then taking away all the clauses whose size exceed r.
- $\overline{\wedge}$: Let $\{f^C_{1,Y}\}_{Y \in \mathcal{N}(k)}$ and $\{f^C_{2,Y}\}_{Y \in \mathcal{N}(k)}$ be two approximators, each being sets of monotone CNF formulas. The approximate AND of these approximators, denoted $\{f^C_{1,Y}\}_{Y \in \mathcal{N}(k)} \overline{\wedge} \{f^C_{2,Y}\}_{Y \in \mathcal{N}(k)}$, is defined as the set of monotone DNF

formulas, denoted $\{f^D_Y\}_{Y\in\mathcal{N}(k)}$, where, for each $Y \in \mathcal{N}(k)$, f^D_Y is defined as follows:

1. For each $Y_1, Y_2 \in \mathcal{N}(k)$, let $f^D_{Y_1,Y_2}$ be the DNF formula obtained from the CNF formula $f^C_{Y_1} \wedge f^C_{Y_2}$ by transforming into the equivalent monotone DNF formula and then taking away all the clauses whose size exceed l.
2. For each $Y \in \mathcal{N}(k)$, set f'^D_Y to

$$f'^D_Y = \bigvee_{Y_1,Y_2:Y=Y_1\cup Y_2} f^D_{Y_1,Y_2}.$$

3. For each $Y \in \mathcal{N}(k)$, let f^D_Y be the monotone DNF formula obtained by replacing each term $\wedge L$ of f'^D_Y with the term $\wedge\lceil\lfloor L\rfloor\rceil$.

In what follows, an approximator $\{f_Y\}_{Y\in\mathcal{N}(k)}$ will be simply denoted by $\{f_Y\}_Y$. Note that $(\{f^D_{1,Y}\}_Y \overline{\vee} \{f^D_{2,Y}\}_Y) \geq (\{f^D_{1,Y}\}_Y \vee \{f^D_{2,Y}\}_Y)$ holds for any approximators $\{f^D_{1,Y}\}_Y$ and $\{f^D_{2,Y}\}_Y$ and that $(\{f^C_{1,Y}\}_Y \overline{\wedge} \{f^C_{2,Y}\}_Y) \leq (\{f^C_{1,Y}\}_Y \wedge \{f^C_{2,Y}\}_Y)$ holds for any approximators $\{f^C_{1,Y}\}_Y$ and $\{f^C_{2,Y}\}_Y$.

Now that the approximate operations are defined for nonmonotone circuits, approximators computed by gates are determined from the bottom to the top. In order to get approximators determined in this way, we need to specify the approximator corresponding to input variables: The approximator corresponding to x_i is simply defined to be x_i itself.

The proof of Theorem 4.2 proceeds in a similar way to those in the previous section. We first prove that the total error in an approximator circuit is large (Lemma 4.4) and then prove that the local error at any single gate is small (Lemmas 4.5 and 4.6). Using these lemmas, the theorem follows from a simple calculation; let $s = m^c$, $l = m^{c/2}/4$, $k = m^{c/2-\epsilon}$, and $r = m^{1-c}/4$.

LEMMA 4.4. *Let $\overline{f}$ be a function computed by an approximator circuit. Then $\overline{f}$ is identically* 0 *or the number of bad mappings for which $\overline{f}$ outputs* 1 *on the corresponding bad vector is at least* $(1/2)(1/2s)^k$.

Proof. If $\overline{f}$ is identically 0, then the first conclusion of the lemma holds. If not, because of the assumption that the output gate of an approximator circuit is $\wedge$ gate, $\overline{f} \geq \wedge Y\lceil W\rceil$ for some $Y \in \mathcal{N}(k)$ and for some $W \subseteq V$ with $|W| \leq l$. Then a bad mapping ϕ satisfying the following conditions makes the approximator $\overline{f}$ take the value 1 : (i) For every variable $y \in Y$, at least one endpoint of y has value 0 on the first element of ϕ, and (ii) the first elements of $\phi(v)$ for $v \in W$ are all distinct and none of them are 0. The number of such bad mappings is at least

$$\begin{aligned}
\binom{s-1}{l}\left(\frac{m}{s}\right)^l\binom{m/s}{k}(m-k-l)! &= m!\frac{\binom{s-1}{l}\left(\frac{m}{s}\right)^l\binom{m/s}{k}(m-k-l)!}{m!} \\
&\geq \prod_{i=0}^{l-1}\left(\frac{(m/s)(s-1-i)!}{m-i}\right)\prod_{i=0}^{k-1}\left(\frac{m/s-i}{m-k-i}\right) \\
&\geq \left(\frac{(m/s)(s-l)}{m-l+1}\right)^l\left(\frac{m/s-k+1}{m-l-k+1}\right)^k \\
&\geq \left(\frac{(m/s)(s-l)}{m}\right)^l\left(\frac{m/s-k+1}{m-l-k+1}\right)^k \\
&\geq \left(1-\frac{l}{sm}\right)^l\left(\frac{m/s-k+1}{m-l-k+1}\right)^k
\end{aligned}$$

$$\geq \left(1-\frac{1}{m}\right)^{l}\left(\frac{m/s-k+1}{m-l-k+1}\right)^{k} \geq \frac{1}{2}\left(\frac{1}{2s}\right)^{k}.$$

This completes the proof of Lemma 4.4. □

LEMMA 4.5. *Suppose that an $\vee$ gate and an $\overline{\vee}$ gate are given as input the same pair of approximators. Then the number of bad mappings for which the $\vee$ and the $\overline{\vee}$ gates produce different outputs on the corresponding bad vectors is at most*

$$m^{2k}(m/2)^{r+1}(m-r-1)!.$$

Proof. Suppose that an $\vee$ gate and an $\overline{\vee}$ gate are given as input the same pair of approximators, denoted $\{f^D_{1,Y}\}_Y$ and $\{f^D_{2,Y}\}_Y$, respectively. Let $\{f^D_{1,Y}\}_Y \overline{\vee} \{f^D_{2,Y}\}_Y$ be denoted by $\{f^C_Y\}_Y$.

Let $Y \in \mathcal{N}(k)$ be fixed arbitrarily. We first estimate the number of bad mappings for which $f^D_{1,Y} \vee f^D_{2,Y}$ outputs 0 and f^C_Y outputs 1 on the corresponding bad vectors by following the arguments in the proof of Lemma 3.4. The condition $l \leq \sqrt{s-1}/2$ in the proof of Lemma 3.4 is satisfied. Thus we can estimate the number of such bad mappings by using the same argument to the proof of the lemma except for replacing all the formulas $(s-1)$ in that proof by s. This modification is needed since the size of the ranges of the first components of the bad mappings is changed from $s-1$ to s. Thus we can verity that, for each $Y \in \mathcal{N}(k)$, the number of bad mappings for which $f^D_{1,Y} \vee f^D_{2,Y}$ outputs 0 and f^C_Y outputs 1 on the corresponding bad vectors is at most $(m/2)^{r+1}(m-r-1)!$. Thus, since the number of elements in $|\mathcal{N}(k)|$ is at most m^{2k}, we obtain the desired bound. □

LEMMA 4.6. *Suppose that an $\wedge$ gate and an $\overline{\wedge}$ gate are given as input the same approximators. Then the number of good vectors for which the $\wedge$ and the $\overline{\wedge}$ gates produce different outputs is at most*

$$\frac{m^{4k}(m/2)^{l+1}(m-l-1)!}{s!(m-s)!}.$$

Proof. Suppose that an $\wedge$ gate and an $\overline{\wedge}$ gate are given as input the same approximators, denoted $\{f^C_{1,Y}\}_Y$ and $\{f^C_{2,Y}\}_Y$, respectively. Let $\{f^D_Y\}_Y = \{f^C_{1,Y}\}_Y \overline{\wedge} \{f^D_{2,C}\}_Y$. It is easy to observe that steps 2 and 3 in the definition of the approximator $\overline{\wedge}$ introduce no error on good vectors. Thus we have only to estimate the amount of errors on good vectors introduced by replacing the product of two CNF formulas $f^C_{1,Y_1} \wedge f^C_{2,Y_2}$ by a DNF formula $f^D_{Y_1,Y_2}$ for each $Y_1, Y_2 \in \mathcal{N}(k)$.

Consider $Y_1, Y_2 \in \mathcal{N}(k)$ to be arbitrarily fixed. We estimate the number of good vectors for which $f^C_{1,Y_1} \wedge f^C_{2,Y_2}$ outputs 1 and $f^D_{Y_1,Y_2}$ outputs 0 by the argument of the proof of Lemma 3.4. Note that the condition $r \leq m/(4s)$ in the proof of Lemma 3.4 holds. Thus we can verify that the number of such good vectors is at most $(m/2)^{l+1}(m-l-1)!/\{(s!(m-s)!\}$ by the same argument as in the proof of Lemma 3.4. Thus, since the number of choices of Y_1 and Y_2 is at most $|\mathcal{N}(k)|^2 \leq m^{4k}$, we obtain the desired bound. □

We now proceed to the proof of Theorem 4.2. Instead of proving Theorem 4.2, we prove Theorem 4.7, which claims a stronger statement. Intuitively, Theorem 4.7 says that the condition "$\mathrm{Min}_k(C)$ is identical to $\mathrm{Min}(\mathrm{CLIQUE}(m,s))$," which appears in Theorem 4.2, can be replaced by the somewhat weaker condition, "the cardinality of $\mathrm{Min}_k(C)$ is not too small."

THEOREM 4.7. *Let c and ϵ be constants such that $0 < c \leq 2/3$ and $0 < \epsilon < c/2$. Suppose that a circuit C computes the function CLIQUE(m, m^c) and suppose that $h = h(m) = o(m^{c/2-\epsilon})$. If*

$$|\mathrm{Min}_{m^{c/2-\epsilon}}(C)| > \frac{1}{m^h}|\mathrm{Min}(\mathit{CLIQUE}(m, m^c))|$$

holds, then the size of C is $\exp(\Omega(m^{c/2}))$.

Proof. Recall that $s = m^c$, $l = m^{c/2}/4$, $k = m^{c/2-\epsilon}$ and $r = m^{1-c}/4$. Let C be a circuit which computes the function CLIQUE(m, s) and let $\overline{f}$ be the function computed by the approximator circuit of the monotone analogue of the circuit C. Suppose that $h = h(m) = o(m^c)$ and $|\mathrm{Min}_k(C)| > \frac{1}{m^h}|\mathrm{Min}_k(\mathrm{CLIQUE}(m, s))| = \frac{1}{m^h}\frac{m!}{s!(m-s)!}$. By Fact 4.3 and Lemmas 4.5 and 4.6, the size of the circuit C is at least

$$\min\left(\frac{m!}{m^h m^{4k}(m/2)^{l+1}(m-l-1)!}\ ,\ \frac{m!}{2(2s)^k m^{2k}(m/2)^{r+1}(m-r-1)!}\right).$$

This is lower bounded by

$$\min\left(\frac{(m-l)^{l+1}}{m^{5k}(m/2)^{l+1}}\ ,\ \frac{(m-r)^{r+1}}{m^{3k}(m/2)^{r+1}}\right) = \frac{1}{m^{5k}}\exp(\Omega(\min(l, r)))$$
$$= \exp(\Omega(m^{c/2} - 5k\log m)) = \exp(\Omega(m^{c/2})).$$

The second equality holds since $1 - c \geq c/2$ if $c \leq 2/3$. This completes the proof of Theorem 4.7. □

Before closing this section we comment on how our results relate to Razborov's results [11] pointing out the limitations of the approximation method. His result says that $\omega(n^2)$ lower bounds for nonmonotone circuits could not be obtained using the approximation method. Our results, which establish exponential lower bounds for nonmonotone circuits, apparently conflict with the results due to Razborov. From these conflicting results we may conclude that the amount of negations in the circuits we deal with in this paper is too small to make Razborov's arguments valid. It is an interesting open problem to show a bound on the amount of negation beyond which we have to go to make Razborov's arguments valid.

Acknowledgment. We would like to thank anonymous referees for making many suggestions for improving the presentation of this paper.

REFERENCES

[1] N. ALON AND R. B. BOPPANA, *The monotone circuit complexity of Boolean functions*, Combinatorica, 7 (1987), pp. 1–22.

[2] A. ANDREEV, *On a method for obtaining lower bounds for the complexity of individual monotone functions*, Soviet Math. Doklady, 31 (1985), pp. 530–534.

[3] C. BERG AND S. ULFBERG, *Symmetric approximation arguments for monotone lower bounds without sunflowers*, Comput. Complexity, 8 (1999), pp. 1–20.

[4] A. HAKEN, *Counting bottlenecks to show monotone $P \neq NP$*, in Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 36–40.

[5] D. HARNIK AND R. RAZ, *Higher lower bounds on monotone size*, in Proceedings of the 32nd Annual Symposium on Theory of Computing, Portland, OR, 2000, pp. 378–387.

[6] S. JUKNA, *Finite limits and monotone computations: The lower bounds criterion*, in Proceedings of the 12th IEEE Conference on Computational Complexity, Ulm, Germany, 1997, pp. 302–313.

[7] M. Karchmer, *On proving lower bounds for circuit size*, in Proceedings of the 8th Structure in Complexity Theory, San Diego, CA, IEEE Computer Science Press, Los Alamos, CA, 1993, pp. 112–118. Also in Feasible Mathematics II, P. Clote and J. Remmel, eds., Birkhaüser Boston, Boston, MA, 1995, pp. 245–255.

[8] K. Nakayama and A. Maruoka, *Loop circuits and their relation to Razborov's approximation model*, Inform. and Comput., 119 (1995), pp. 154–159.

[9] A. A. Razborov, *Lower bounds on the monotone complexity of some boolean functions*, Soviet Math. Doklady, 31 (1985), pp. 354–357.

[10] A. A. Razborov, *On the method of approximations*, in Proceedings of the 21st Annual Symposium on Theory of Computing, Seattle, WA, 1989, pp. 167–176.

[11] A. A. Razborov and S. Rudich, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.

[12] J. Simon and S. C. Tsai, *On the bottleneck counting argument*, Theoret. Comput. Sci., 237 (2000), pp. 429–437.

[13] A. Wigderson, *The fusion method for lower bounds in circuit complexity*, in Paul Erdős Is Eighty, Vol. 1, Keszthely, Hungary, Boklai Soc. Math. Stud., 1993, pp. 453–468.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 448–461

ALGORITHMS FOR DATA MIGRATION WITH CLONING*

SAMIR KHULLER†, YOO-AH KIM‡, AND YUNG-CHUN (JUSTIN) WAN‡

Abstract. Our work is motivated by the problem of managing data on storage devices, typically a set of disks. Such high-demand storage servers are used as Web servers or as multimedia servers for handling high demand for data. As the system is running, it needs to dynamically respond to changes in demand for different data items. In this work we study the *data migration problem*, which arises when we need to quickly change one storage configuration into another. We show that this problem is NP-hard. In addition, we develop polynomial-time approximation algorithms for this problem and prove a worst-case bound of 9.5 on the approximation factor achieved by our algorithm.

Key words. approximation algorithms, multicast, broadcast, gossip

AMS subject classifications. 68W25, 68Q25, 05C15

DOI. 10.1137/S009753970342585X

1. Introduction.

To handle high demand, especially for multimedia data, a common approach is to replicate data objects within the storage system. Typically, a large storage server consists of several disks connected by a dedicated network, called a *storage area network*. Disks typically have constraints on storage as well as the number of clients that can access data from a single disk simultaneously. With the recent interest in autonomic computing,[1] a relevant goal is to have the system automatically respond to changes in demand patterns and to recompute data layouts.

Approximation algorithms have been developed [24, 25, 9, 17] to map known demand for data to a specific data layout pattern to maximize utilization, where the utilization is the total number of clients that can be assigned to a disk that contains the data they want. In the layout, we compute not only how many copies of each item we need but also a layout pattern that specifies the precise subset of items on each disk. The problem is NP-hard, but there are polynomial-time approximation schemes [9, 25, 17]. Given the relative demand for data, the algorithm computes an almost optimal layout.

Over time as the demand for data changes, the system needs to create *new* data layouts. The problem we are interested in is the problem of computing a data migration plan for the set of disks to convert an initial layout to a target layout. We assume that data objects have the same size (these could be data blocks, or files) and that it takes the same amount of time to migrate any data item from one disk to another disk. The crucial constraint is that each disk can participate in the transfer of only one item—either as a sender or as a receiver. Our goal is to find a migration schedule to minimize the time taken to complete the migration (makespan).

A special case of this was studied by Hall et al. [11]—they compute a movement schedule, but this *does not allow* the creation of new copies of any data object. It

*Received by the editors April 15, 2003; accepted for publication (in revised form) December 9, 2003; published electronically February 24, 2004. A preliminary version of this paper appeared in *Proceedings of the ACM Symposium on Principles of Database Systems*, 2003, pp. 27–36. This research was supported by NSF awards CCR-9820965 and CCR-0113192.

http://www.siam.org/journals/sicomp/33-2/42585.html

†Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 (samir@cs.umd.edu).

‡Department of Computer Science, University of Maryland, College Park, MD 20742 (ykim@cs.umd.edu, ycwan@cs.umd.edu).

[1] http://www.research.ibm.com/autonomic/

addresses only the data *movement* problem. (So, for example, one cannot create extra copies of any data item, but can only change which disks they are stored on.) The problem they studied is formally defined as follows: Given a set of disks, with each storing a subset of items and a specified set of move operations (each move operation specifies which data object needs to be moved from one disk to another), how do we schedule these move operations? If there are no storage constraints, then this is exactly the problem of edge coloring the following multigraph. Create a graph that has a node corresponding to each disk, and a directed edge corresponding to each move operation that is specified. Algorithms for edge-coloring multigraphs can now be applied to produce a migration schedule since each color class represents a matching in the graph that can be scheduled simultaneously. Computing a solution with the minimum number of rounds is NP-hard, but several good approximation algorithms are available for edge coloring. With space constraints on the disk, the problem becomes challenging. Hall et al. [11] showed that with the assumption that each disk has one spare unit of storage, very good constant factor approximations can be developed. The algorithms use at most $4\lceil\Delta_G/4\rceil$ colors with at most $n/3$ bypass nodes, or at most $6\lceil\Delta_G/4\rceil$ colors without bypass nodes, where Δ_G is the maximum degree of a node and n is the number of nodes. Note that a bypass node is a node that is not the target of a move operation but is used as an intermediate holding point for a data item.

On the other hand, to handle high demand for popular objects, new copies will have to be dynamically created and stored on different disks. This means that we crucially need the ability to have a "copy" operation in addition to "move" operations. In fact, one of the crucial lower bounds used in the work on data migration [11] is based on a degree property of the multigraph. For example, if the maximum degree of a node is δ, then this is a lower bound on the number of rounds that are required, since in each round at most one transfer operation involving this node may be done. For copying operations, clearly this lower bound is not valid. For example, suppose we have a single copy of a data item on a disk. Suppose we wish to create δ copies of this data item on δ distinct disks. Using the transfer graph approach, we could specify a "copy" operation from the source disk to each of the δ disks. Notice that this would take at least δ rounds. However, by using newly created copies as additional sources, we can create δ copies in $\lceil\log(\delta+1)\rceil$ rounds, as in the classic problem of broadcasting by using newly created copies as sources for the data object. (Essentially each copy spawns a new copy in each round.)

The most general problem of interest is the *data migration problem with cloning* when data item i resides in a specified (source) subset S_i of disks and needs to be moved to a (destination) subset D_i. In other words, each data item that initially belongs to a subset of disks needs to be moved to another subset of disks. (We might need to create new copies of this data item and store it on an additional set of disks.) See Figure 1.1 for an example. If each disk had exactly one data item and needs to copy this data item to every other disk, then it is exactly the problem of gossiping. We show that this problem is NP-hard by reduction from edge coloring and develop a polynomial-time 9.5-approximation algorithm for it. For all our algorithms, we move data only to disks that need the data. Thus we use no bypass nodes. The total number of data transfers performed is thus the minimum possible. In addition, we have implemented our algorithm and compared its performance to several other heuristics [10].

Different communication models can be considered based on how the disks are connected. We use the same model as in the work by Hall et al. [11] and Anderson

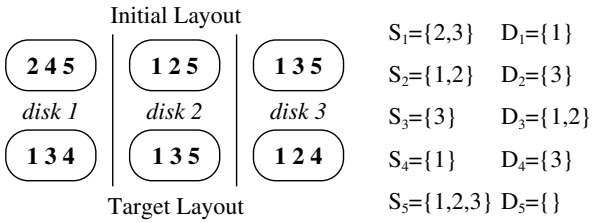


FIG. 1.1. *An initial and target layout and their corresponding S_i's and D_i's.*

et al. [1] where the disks may communicate on any matching; in other words, the underlying communication graph allows for communication between any pair of devices via a matching (a switched storage network with unbounded backplane bandwidth). Moreover, to model the limited switching capacity of the network connecting the disks, one could allow for choosing any matching of bounded size as the set of transfers that can be done in each round. We call this a *bounded-size matching model*. Using the constant factor approximation algorithm for the unbounded matching model, we can also develop a constant factor approximation algorithm for the bounded-size matching model.

One interesting generalization would be for the situation when clusters of disks are connected in a wide area network. The time required to transfer one unit of data between a pair of disks in different clusters may be an order of magnitude higher than the time required to transfer data between a pair of disks in the same cluster. We can model this by a communication graph model where the number of rounds required to transfer one unit of data between a pair of disks in different clusters is a certain number of rounds, and one round is required to transfer one unit of data between a pair of disks in the same cluster.

1.1. Relationship to gossiping and broadcasting. The problems of gossiping and broadcasting have been the subject of extensive study [19, 13, 15, 3, 4, 16, 7, 8]. These play an important role in the design of communication protocols in various kinds of networks. The *gossip problem* is defined as follows: There are n individuals. Each individual has an item of gossip that they wish to communicate to everyone else. Communication is typically done in rounds, where in each round an individual may communicate with at most one other individual. Some communication models allow for the full exchange of all items of gossip known to each individual in a single round. Other models allow the sending of only one item of gossip from one to the other (half-duplex) or allow each individual to send an item to the individual they are communicating with in this round (full-duplex). In addition, there may be a communication graph whose edges indicate which pairs of individuals are allowed to communicate directly in each round. (In the classic gossip problem, also called the telephone model, communication may take place between any pair of individuals; in other words, the communication graph is the complete graph.) In the *broadcast problem*, one individual needs to convey an item of gossip to every other individual. The two parameters typically used to evaluate the algorithms for this problem are (i) the number of communication rounds, and (ii) the total number of telephone calls placed.

The problems we study are generalizations of the above-mentioned gossiping and broadcasting problems. The basic generalizations we are interested in are of three kinds: (a) each item of gossip needs to be communicated to only a subset of

individuals; (b) several items of gossip may be known to an individual; and (c) a single item of gossip can initially be shared by several individuals.

The communication model we use is the half-duplex model, where only one item of gossip may be communicated between two communicating individuals during a single round. Each individual may communicate (either send or receive an item of data) with at most one other individual in a round. This model best captures the connection of parallel storage devices that are connected on a network and is most appropriate for our application. This model is one of the most widely used in all the work related to gossiping and broadcasting.

1.2. Contributions and outline of paper. In section 2 we define the basic model of communication and the notation used in the paper.

This is the first work that relates broadcasting and gossiping with the data migration problem. In section 3 we develop a 9.5-approximation algorithm for the general data migration problem. This is the first approximation algorithm for this problem. While two of the lower bounds used are quite simple, we develop a new lower bound using network flows and use this in the algorithm. (Without this lower bound, the best bound we can obtain is a $O(\log n)$ factor.) We show the data migration problem is NP-hard at the end of section 3.

In section 4 we develop a constant factor approximation algorithm for the bounded-size matching model.

1.3. Other related work. The paper by Liben-Nowell [21] considers a problem very similar to multisource multicast, which is exactly the data migration problem with restrictions that each disk contain at most one source item and that each item have at most one source. However, the model used is different than the one that we use. In that model, in each telephone call, a pair of users can exchange all the items of gossip that they know. The objective is to simply minimize the total number of phone calls required to convey item i of gossip to set D_i of users. In our case, since each item of gossip is a data item that might take considerable time to transfer between two disks, we cannot assume that an arbitrary number of data items can be exchanged in a single round. Several other papers use the same telephone call model [2, 6, 12, 16, 28].

Other related problems that have been studied are the set-to-set gossiping problem [20, 23], where we are given two possibly intersecting sets A and B of gossipers and the goal is to minimize the number of calls required to inform all gossipers in A of all the gossip known to members in B. Liben-Nowell [21] generalizes this work by defining for each gossiper i the set of relevant gossip that they need to learn. This is just like our multisource multicast problem when the number of items is equal to the number of disks, except that the communication model is different, as well as the objective function.

We have also studied several special cases of the data migration problem with cloning, where each data item has only one copy initially, and have developed algorithms with better performance guarantees [18]. One special case we studied is the multisource multicast problem. We showed that this problem is NP-hard by a reduction from a restricted version of 3SAT and gave a polynomial-time algorithm with approximation ratio of $3 + o(1)$ using a simplified version of the algorithm developed in this paper.[2] Allowing bypass nodes, we improved the approximation ratio to 3.

[2]The work in [18] presents a 4-approximation algorithm. The improvement to $3 + o(1)$ can be obtained from the full version of [18], available at http://www.cs.umd.edu/projects/smart/papers/multicast.pdf.

Another special case is the *multisource broadcast* problem, which is the same as the multisource multicast problem except that all disks demand all items; i.e., D_i is all the disks minus its source. We developed a polynomial-time algorithm using at most 3 more rounds than the optimal. The last special case we studied is the *single-source multicast* problem, when Δ data items, each having only one copy, are all stored on the same disk, and data item i needs to be sent to a specified subset D_i of disks. We developed a polynomial-time algorithm using at most Δ more rounds than the optimal. Farley [8] solved a simpler case optimally, namely, the *single-source broadcast* problem, where Δ data items are stored on a single disk, and all items need to be broadcast to all disks.

2. Models and definitions. In the *data migration problem*, we have N disks and Δ data items. For each item i, there is a subset of disks S_i and D_i. Initially only the disks in S_i have item i, and all disks in D_i want to receive i. Note that after a disk in D_i receives item i, it can be a source of item i for other disks in D_i that have not received the item as yet. Our goal is to find a migration schedule using the minimum number of rounds, that is, to minimize the total amount of time to finish the schedule. We assume that the underlying network is fully connected and the data items are all the same size; in other words, it takes the same amount of time to migrate an item from one disk to another. The crucial constraint is that each disk can participate in the transfer of only one item—either as a sender or receiver. Moreover, as we do not use any bypass nodes, all data is sent only to disks that desire it.

Our algorithms make use of known results on edge coloring of multigraphs. Given a graph G with max degree Δ_G and multiplicity μ, the following results are known (see Bondy and Murty [5], for example). Let χ' be the edge chromatic number of G.

THEOREM 2.1 (Vizing [29]). *If G has no self-loops, then $\chi' \leq \Delta_G + \mu$.*

THEOREM 2.2 (Shannon [26]). *If G has no self-loops, then $\chi' \leq \lfloor \frac{3}{2}\Delta_G \rfloor$.*

3. The data migration algorithm. Define β_j as $|\{i | j \in D_i\}|$, i.e., the number of different sets D_i to which a disk j belongs. We then define β as $\max_{j=1,\ldots,N} \beta_j$. In other words, β is an upper bound on the number of items a disk may need. Note that β is a lower bound on the optimal number of rounds, since the disk j that attains the maximum needs at least β rounds to receive all the items i such that $j \in D_i$, since it can receive at most one item in each round.

Moreover, we may assume that $D_i \neq \emptyset$ and $D_i \cap S_i = \emptyset$. This is because we can define the destination set D_i as the set of disks that need item i and do not currently have it.

Since the algorithm is somewhat complex, we first give a high-level description of the algorithm and then discuss the various steps in the following lemmas. Dealing with multiple data items sharing common disks causes some difficulty.

DATA MIGRATION ALGORITHM.

1. For an item i decide a source $s_i \in S_i$ so that $\alpha = \max_{j=1,\ldots,N}(|\{i | j = s_i\}| + \beta_j)$ is minimized. In other words, α is the maximum number of items for which a disk may be a source (s_i) or destination. *Note that α is also a lower bound on the optimal number of rounds.* In Lemma 3.1 we will show how we can find a source for each item.
2. Find a transfer graph for items that have $|D_i| \geq \beta$ as follows:
   (a) We first compute a disjoint collection of subsets G_i, $i = 1, \ldots, \Delta$. Moreover, $G_i \subseteq D_i$ and $|G_i| = \lfloor \frac{|D_i|}{\beta} \rfloor$. Figure 3.1(a) shows an example of choosing sets G_i. (In Lemma 3.2, we will show how such G_i's can be obtained.)

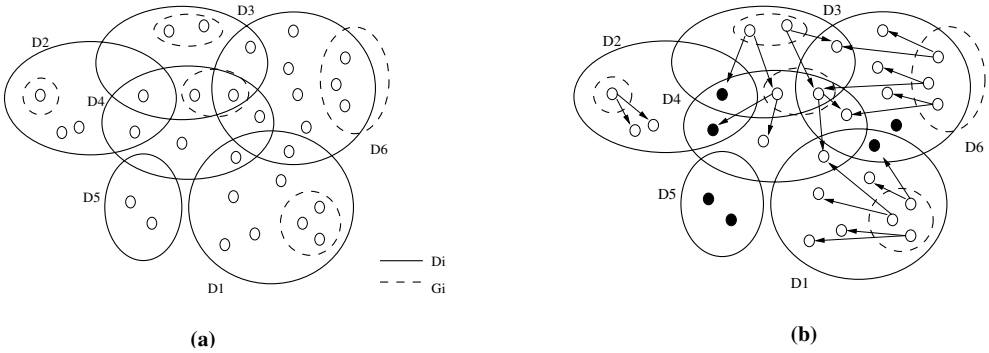


FIG. 3.1. (a) *An example of choosing G_i in step* 2(a)*, where $\Delta = 6$ and $\beta = 3$.* (b) *Transfer graph constructed in step* 2(c)*. Disks marked as black do not receive some data items and will be taken care of in step* 3.

(b) We have each item i sent to the set G_i as shown in Lemma 3.5.
(c) We create a transfer graph as follows. Each disk is a node in the graph. We add directed edges from disks in G_i to $(\beta-1)\lfloor\frac{|D_i|}{\beta}\rfloor$ disks in $D_i \setminus G_i$ such that the out-degree of each node in G_i is at most $\beta - 1$ and the in-degree of each node in $D_i \setminus G_i$ from G_i is 1. Figure 3.1(b) shows an example of the transfer graph constructed in this step. We redefine D_i as the set of $|D_i \setminus G_i| - (\beta-1)\lfloor\frac{|D_i|}{\beta}\rfloor$ disks which do not receive item i so that they can be taken care of in step 3. Note that the redefined set D_i has size $< \beta$.

3. Find a transfer graph for items such that $|D_i| < \beta$ as follows:
   (a) For each item i, find a new source s_i' in D_i. A disk j can be a source s_i' for several items as long as $\sum_{i\in I_j} |D_i| \le 2\beta - 1$, where I_j is a set of items for which j is a new source. See Lemma 3.7 for the details of this step.
   (b) Send each item i from s_i to s_i'.
   (c) Create a transfer graph. We add a directed edge from s_i' to all disks in $D_i \setminus \{s_i'\}$. Lemma 3.9 will show that the out-degree of a disk does not exceed $2\beta - 4$.
4. We now find an edge coloring of the transfer graph obtained by merging two transfer graphs in steps 2(c) and 3(c). The number of colors used is an upper bound on the number of rounds required to ensure that each disk in D_i gets item i. In Lemma 3.10 we derive an upper bound on the number of required colors.

LEMMA 3.1. *We can find a source $s_i \in S_i$ for each item i so that* $\max_{j=1,\ldots,N}(|\{i|j = s_i\}| + \beta_j)$ *is minimized, using a flow network.*

Proof. We create a flow network with a source s and a sink t as shown in Figure 3.2. We have two sets of nodes corresponding to disks and items. Add directed edges from s to nodes for items and also directed edges from item i to disk j if $j \in S_i$. The capacities of all these edges is one. Finally we add an edge from the node corresponding to disk j to t with capacity $\alpha - \beta_j$. We want to find the minimum α so that the maximum (integral) flow of the network is Δ. We can do this by checking if there is a flow of Δ with α starting from $\max \beta_j$ and increasing by one until it is satisfied. If there is outgoing flow from item i to disk j, then we set j as s_i. ∎

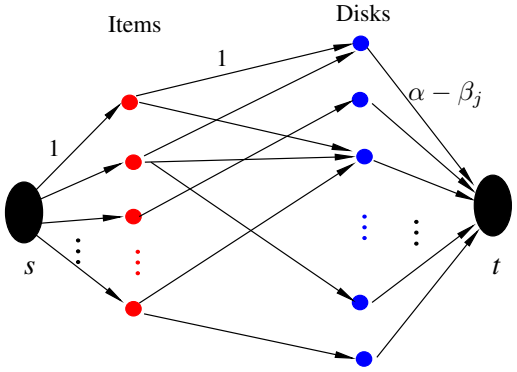


FIG. 3.2. *Flow network to find* α.

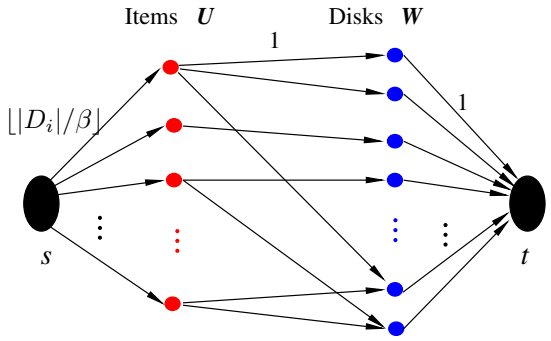


FIG. 3.3. *Flow network to find* G_i.

LEMMA 3.2. *There is a way to choose disjoint sets* G_i *for each* $i = 1, \ldots, \Delta$ *such that* $|G_i| = \lfloor \frac{|D_i|}{\beta} \rfloor$ *and* $G_i \subseteq D_i$.

Proof. First note that the total size of the sets G_i is at most N.

$$\sum_{i=1}^{\Delta} |G_i| \leq \sum_{i=1}^{\Delta} \frac{|D_i|}{\beta} = \frac{1}{\beta} \sum_{i=1}^{\Delta} |D_i|.$$

Note that $\sum_{i=1}^{\Delta} |D_i|$ is at most βN by definition of β. This proves the upper bound of N on the total size of all sets G_i.

We now show how to find the sets G_i. As shown in Figure 3.3, we create a flow network with a source s and sink t. In addition we have two sets of vertices U and W. The first set U has Δ nodes, each corresponding to an item. The set W has N nodes, each corresponding to a disk in the system. We add directed edges from s to each node in U such that the edge (s, i) has capacity $\lfloor \frac{|D_i|}{\beta} \rfloor$. We also add directed edges with unit capacity from node $i \in U$ to $j \in W$ if $j \in D_i$. We add unit capacity edges from nodes in W to t. We find a max-flow from s to t in this network. The min-cut in this network is obtained by simply selecting the outgoing edges from s. To see this, note that we can find a fractional flow of this value as follows: saturate all the outgoing edges from s. From each node i there are $|D_i|$ edges to nodes in W. Suppose $\lambda_i = \lfloor \frac{|D_i|}{\beta} \rfloor$. Send $\frac{1}{\beta}$ units of flow along $\lambda_i \beta$ outgoing edges from i. Note that

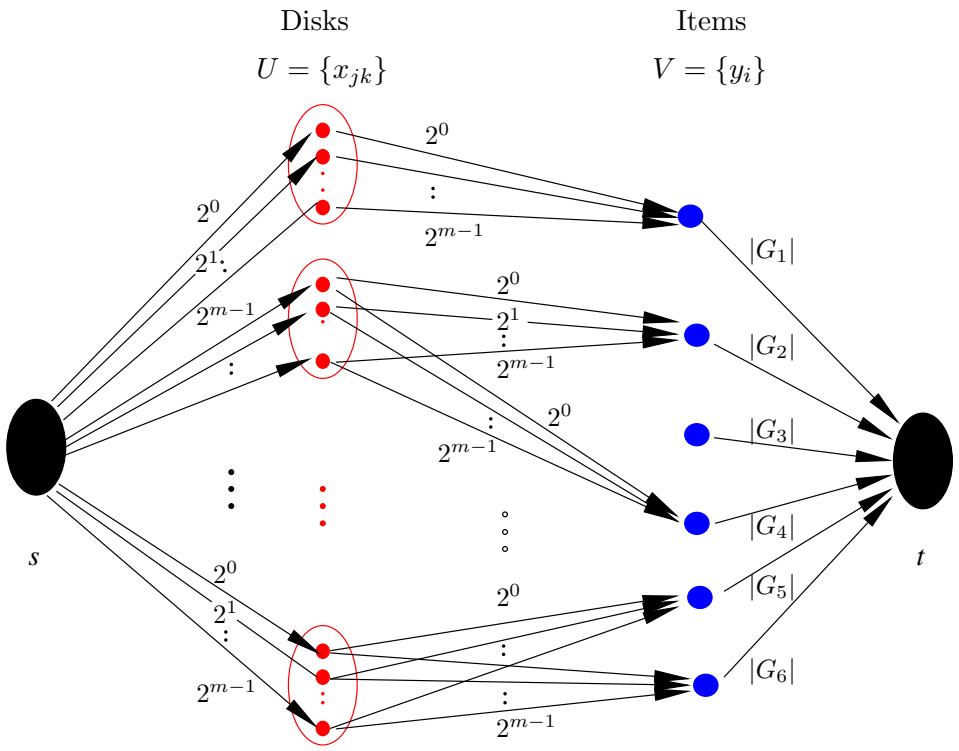


FIG. 3.4. *An example of constructing F_m where $\Delta = 6$.*

since $\lambda_i \beta \leq |D_i|$ this can be done. Observe that the total incoming flow to a vertex in W is at most 1 since there are at most β incoming edges, each carrying at most $\frac{1}{\beta}$ units of flow. An integral max-flow in this network will correspond to $|G_i|$ units of flow going from s to i and from i to a subset of vertices in D_i before reaching t. The vertices to which i has nonzero flow will form the set G_i. □

For step 2(b), a simple solution would be to broadcast the data to each group G_i from the chosen source, since the groups are disjoint. The only thing we have to be careful of is that the sources for many data items are shared. However, this broadcast takes at least $\max_i \log |G_i|$ rounds. Unfortunately, we cannot argue that this is a valid lower bound since even though D_i is large, if S_i is large, then there could be a solution using $O(1)$ rounds. This would give us an $O(\log N)$ approximation guarantee. The method described below develops stronger lower bounds for this situation.

Let M be the number of steps required to send all items i to all disks in G_i in an optimal schedule of step 2(b). To find a lower bound for M, we construct the following flow network F_m (parameterized by an integer m) as shown in Figure 3.4. We have a source s and two sets of nodes U and V. U has $N \cdot m$ nodes x_{jk} ($j = 1, \ldots, N$, $k = 1, \ldots, m$). V has Δ nodes y_i ($i = 1, \ldots, \Delta$) and y_i has demand $|G_i|$. There is an edge e_{ijk} from x_{jk} to y_i and its capacity c_{ijk} is 2^{m-k} if a disk j has item i initially. There are edges from s to nodes x_{jk} in U with capacity 2^{m-k}.

LEMMA 3.3. *If m' is the smallest number such that we can construct a solution of $F_{m'}$ that satisfies all demands $|G_i|$, then $M \geq m'$.*

Proof. Suppose that $M < m'$. Given an optimal schedule of step 2(b), we can construct a solution of the flow network F_M as follows. If a disk j sends item i to

a disk in G_i at round $t \leq M$, which makes f copies in G_i subsequently, we send a flow f from x_{jt} to y_i. Note that f cannot be more than 2^{M-t}, and therefore it does not violate the capacity constraint. Since all disks in G_i receive item i after M rounds with this schedule, the corresponding flow satisfies all demands $|G_i|$. This is a contradiction to the assumption that m' is the smallest number to satisfy all demands $|G_i|$. □

In the solution of the flow network $F_{m'}$, a node x_{jk} may send flow to several nodes. But since in our schedule, a disk can copy only one item to a disk in a round, the solution of the flow in $F_{m'}$ may not correspond to a valid schedule.

LEMMA 3.4. *Given a solution of $F_{m'}$, we can convert it to a solution satisfying the following properties:*

- *Node x_{jk} sends flow to at most one node in V.*
- *The solution satisfies at least $|G_i| - 2^{m'-1}$ demands for each item i.*

Proof. First, we define a variable z_{ijk} for an edge from x_{jk} to y_i and set $z_{ijk} = f_{ijk}/c_{ijk}$, where f_{ijk} is the flow through e_{ijk} in solution $F_{m'}$. We substitute nodes y_{il} $(l = 1, \ldots, \lfloor \sum_{j,k} z_{ijk} \rfloor)$ for each node y_i in V. We distribute edges having nonzero flow to y_i as follows. Sort edges in nonincreasing order of their capacities. Assign edges to y_{i1} until the sum of z values of assigned edges is greater than or equal to one. If the sum is greater than one, we split the last edge (denote as $e_{ij'k'}$) into $e_{ij'k'_1}$ and $e_{ij'k'_2}$. Assign $e_{ij'k'_1}$ to y_{i1} and define $z_{ij'k'_1}$ so that the sum of z values of edges assigned to y_{i1} is exactly one. Set $z_{ij'k'_2} = z_{ij'k'} - z_{ij'k'_1}$. We repeat this so that for all nodes y_{il}, the sums of z values of the assigned edges are one. Let E_{il} be the set of edges assigned to y_{il} and let c_{il}^{max} (c_{il}^{min}) be the maximum (minimum) capacity of the edges in E_{il}. In addition, we denote the edges not assigned to any y_{il} $(l = 1, \ldots, \lfloor \sum_{j,k} z_{ijk} \rfloor)$ as $E_{il'}$ and the maximum capacity of edges in $E_{il'}$ as $c_{il'}^{max}$, where l' is $\lfloor \sum_{j,k} z_{ijk} \rfloor + 1$.

In the resulting bipartite graph with U and $V' = \{y_{il}\}$, z makes a *fractional* matching which matches all vertices in V' but not necessarily all vertices in U. Therefore, we can find an integral matching that matches all vertices in V', and the matching satisfies the first property in the lemma.

Now we merge nodes y_{il} into y_i. Then each y_i matches exactly $\lfloor \sum_{j,k} z_{ijk} \rfloor$ edges. We prove that the sum of capacities of edges matched to y_i is at least $|G_i| - c^{max}$, where c^{max} is the maximum capacity of any edge from U to V, using an analysis similar to that in Shmoys and Tardos [27]. The sum of capacities of edges matched to y_i is at least

$$\begin{aligned}
\sum_{l=1}^{\lfloor \sum_{j,k} z_{ijk} \rfloor} c_{il}^{min} &\geq \sum_{l=2}^{\lfloor \sum_{j,k} z_{ijk} \rfloor + 1} c_{il}^{max} \\
&= \sum_{l=1}^{\lfloor \sum_{j,k} z_{ijk} \rfloor + 1} c_{il}^{max} - c_{i1}^{max} \\
&\geq \sum_{l=1}^{\lfloor \sum_{j,k} z_{ijk} \rfloor + 1} \sum_{e_{ijk} \in E_{il}} c_{ijk} z_{ijk} - c_{i1}^{max} \\
&= \sum_{l=1}^{\lfloor \sum_{j,k} z_{ijk} \rfloor + 1} \sum_{e_{ijk} \in E_{il}} f_{ijk} - c_{i1}^{max} \\
&\geq |G_i| - c^{max}.
\end{aligned}$$

Since $c^{max} \leq 2^{m'-1}$, the second property can be satisfied by setting flow through e_{ijk} as c_{ijk} if e_{ijk} is matched. ∎

LEMMA 3.5. *Step* 2(b) *can be done in* $\alpha + 2m' + 1$ *rounds.*

Proof. We can do this with the following schedule. First we choose $\min(\lfloor \sum_{j,k} z_{ijk} \rfloor + 1, |G_i|)$ disks in G_i and denote those disks as H_i. Disk j sends item i to a disk in H_i if edge e_{ijk} is matched for some k. If $|H_i| > \lfloor \sum_{j,k} z_{ijk} \rfloor$, there is one disk in H_i which cannot receive item i. The disk receives item i from s_i. Then the maximum degree of a disk is at most $m' + \alpha$ and the multiplicity of the undirected version of the transfer graph is 2, since the out-degree of disk j is at most $m' + \alpha - \beta_j$ and the in-degree is at most $\min(\beta_j, 1)$. By Theorem 2.1, it can be done in $m' + \alpha + 2$ rounds.

Now $|H_i|$ nodes in G_i have item i. Since $|G_i|/|H_i| \leq (\sum_{l=1}^{\lfloor \sum_{j,k} z_{ijk} \rfloor} c_{il}^{min} + c^{max})/(\lfloor \sum_{j,k} z_{ijk} \rfloor + 1) \leq c^{max} \leq 2^{m'-1}$, we can make all disks in G_i have item i in an additional $m' - 1$ rounds. ∎

Lemma 3.7 will show how step 3(a) works. The lemma uses the following result from Shmoys and Tardos [27].

THEOREM 3.6 (Shmoys and Tardos [27]). *We are given a collection of jobs* $\mathcal{J}$*, each of which is to be assigned to exactly one machine among the set* $\mathcal{M}$*; if job* $j \in \mathcal{J}$ *is assigned to machine* $i \in \mathcal{M}$*, then it requires* p_{ij} *units of processing time and incurs a cost* c_{ij}*. Suppose that there exists a fractional solution (that is, a job can be assigned fractionally to machines) with makespan* P *and total cost* C*. Then in polynomial time we can find a schedule with makespan* $P + \max p_{ij}$ *and total cost* C*.*

LEMMA 3.7. *For each item* i *we wish to choose a source disk* s_i' *from* D_i*. Let* I_j *be the set of items for which disk* j *is chosen as a source. There is a way to choose the sources such that the following properties hold:*

- *If* $i \in I_j$*, then* $j \in D_i$*.*
- $\sum_{i \in I_j} |D_i| \leq 2\beta - 1$*.*

Proof. We use Theorem 3.6 for this step. For example, we can create an instance of the problem of scheduling machines with costs. Items correspond to jobs and disks correspond to machines. For each item i we define a cost function as follows. $C(i, j) = 1$ if and only if $j \in D_i$; otherwise it is a large constant. The processing time of job i (corresponding to item i) is $|D_i|$ (uniform processing time on all machines). Using Theorem 3.6 [27], the scheduling algorithm finds a schedule that assigns each job (item) to a machine (disk) to minimize the makespan. They show that the makespan is at most the makespan in a fractional solution plus the processing time of the largest job. Moreover, the cost of their solution is at most the cost of the optimal solution, namely, the number of items. We cannot assign an item (job) to a disk (machine) if the disk is not in the destination set for the item.

In our case, it is easy to see that the maximum processing time of any job is $\beta - 1$. We will argue that there is a fractional solution with makespan β. It thus follows that by defining I_j to be the set of items (jobs) assigned to disk (machine) j, the result follows. The fractional solution is obtained by assigning each job fractionally to each machine by setting the assignment variable for job i on machine j to $\frac{1}{|D_i|}$ if $j \in D_i$; then the job is fully assigned fractionally and the fractional load on each machine is at most β. This also gives us a way of finding a fractional solution efficiently. ∎

LEMMA 3.8. *Step* 3(b) *can be done in* $\lfloor \frac{3}{2}\alpha \rfloor$ *rounds.*

Proof. Since disk j can be s_i' in step 3(a) only if $j \in D_i$, $|I_j| \leq \beta_j$. Therefore, a disk j may need to send $\alpha - \beta_j$ items and receive β_j items. That means the maximum degree is α and this transfer can make a multigraph. Given a multigraph with maximum degree Δ_G, we can find an edge coloring using $\lfloor \frac{3}{2}\Delta_G \rfloor$ colors (see

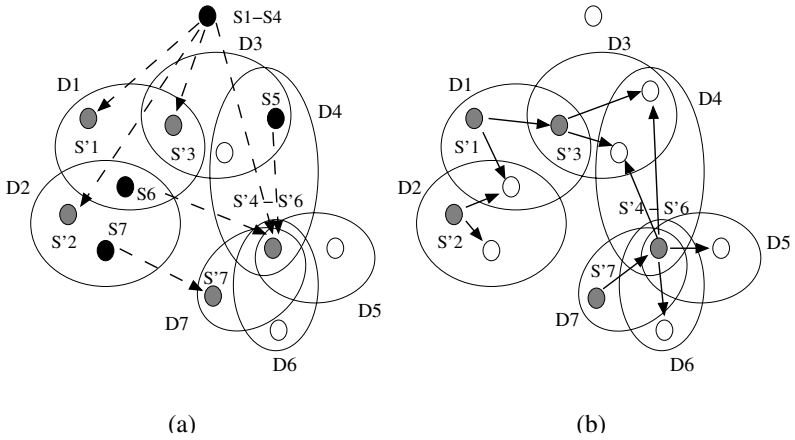


FIG. 3.5. *An example of step* 3 *where* $\alpha = 4$ *and* $\beta = 4$. (a) *Migration from* s_i *to* s'_i. (b) *Migration from* s'_i *to* $D_i \setminus \{s'_i\}$.

Theorem 2.2) and the lemma follows. □

LEMMA 3.9. *The maximum out-degree of a disk in the transfer graph in step* 3(c) *is at most* $2\beta - 4$.

Proof. If disk j is a new source for k items (in other words, $|I_j| = k$), then the out-degree of disk j is $\sum_{i \in I_j} |D_i \setminus \{s'_i\}| = \sum_{i \in I_j} |D_i| - k$.

It is easy to see that the lemma is true for $k \geq 3$ since $\sum_{i \in I_j} |D_i| \leq 2\beta - 1$. For $k = 1$, the lemma is also true since $\sum_{i \in I_j} |D_i| \leq \beta - 1$ and $\beta \geq 2$ (otherwise, there is no set with size less than β). For $k = 2$, $\sum_{i \in I_j} |D_i| \leq 2\beta - 2$, and therefore we have the lemma. □

Figure 3.5 shows an example of migrations in step 3.

LEMMA 3.10. *The number of colors we need for the final transfer graph in step* 4 *is* $(4\beta - 5) + (\beta + 2)$.

Proof. The out-degree of a disk j can be at most $3\beta - 5$ ($\beta - 1$ in step 2(c) and $2\beta - 4$ in step 3(c)). The in-degree is at most β by definition. We claim that the multiplicity of the undirected version of the final transfer graph is $\beta + 2$. Consider all the edges added in step 2(c); we will show the multiplicity induced by these edges is 2. Since all G_i's are disjoint and each disk in G_i sends only item i to disks in $D_i \setminus G_i$, for any pair of disks j_1 and j_2, there can be at most one edge in each direction. Now consider all the edges added in step 3(c); if there is an edge between disk j_1 and disk j_2, no matter which disk is the sender, both disks belong to D_i for some i. Thus, there are at most β edges between j_1 and j_2 since a disk can belong to at most β different D_i's. Therefore, the result follows by Theorem 2.1. □

THEOREM 3.11. *The total number of rounds required for the data migration is at most* $\alpha + 2m' + \lfloor \frac{3}{2}\alpha \rfloor + 5\beta - 2$.

Proof. We need $\alpha + 2m' + 1$ rounds for step 2(b) by Lemma 3.5 and $\lfloor \frac{3}{2}\alpha \rfloor$ rounds for step 3(b) by Lemma 3.8. Migration according to the coloring of the final transfer graph needs $(4\beta - 5) + (\beta + 2)$ by Lemma 3.10. Therefore, we have the theorem. □

COROLLARY 3.12. *Our algorithm is* 9.5*-approximation for the data migration problem.*

Proof. Since α, β, m' are lower bounds for the problem, the corollary follows. □

THEOREM 3.13. *The data migration problem (with copy operation) is NP-hard.*

Proof. We give a reduction from the problem of edge coloring a simple graph with the smallest number of colors (which is known to be NP-hard [14]). Given a graph $G = (V, E)$, we create an instance of a data migration problem with $N = |V|$ disks. For each edge $e_i = (u, v)$ in the graph, we create a new item i, where S_i is $\{u\}$ and D_i is $\{v\}$. It is not difficult to see that the minimum number of colors required in the edge coloring instance is the same as the minimum number of rounds in the corresponding data migration instance. ∎

3.1. A bad example. Here we give an example when our data migration algorithm does not perform very well. Consider the problem where there are Δ source disks, each disk having a separate item; in addition, there are $\Delta - 1$ destination disks, and each disk requests all Δ items. Thus N is equal to $2\Delta - 1$, β is equal to Δ, and $|D_i|$, the number of disks requesting item i, is equal to $\beta - 1$ for all i. In step 3 of our data migration algorithm, we need to find Δ new sources s'_i, but we have only $\Delta - 1$ destination disks. At least one disk d has to be a new source for two items. Therefore step 3(b) takes at least 2 rounds. In disk d, each item in d has to be sent to the remaining $\Delta - 2$ destination disks. The out-degree is exactly $2\Delta - 4$. The in-degree is $\Delta - 2$. So, we have a node of degree $3\Delta - 6$ in the transfer graph, and the total number of rounds is at least $3\Delta - 4$. The optimal strategy is to have $\Delta - 1$ of Δ source disks sending items to destination disks in a round-robin fashion. This method takes only Δ rounds. Therefore, our algorithm cannot perform better than $(3 - \epsilon)$-approximation.

4. Bounded-size matching model. The following algorithm gives a constant factor approximation when at most C transfers are allowed in each round. Let E_i be the transfers in the ith round in the algorithm for the full matching model. Then we split each E_i into $\lceil |E_i|/C \rceil$ sets of size at most C and perform each set in a round.

THEOREM 4.1. *Given ρ-approximation algorithm for the full matching model, we have a $1 + \rho(1 - 1/C)$-approximation algorithm for the bounded-size matching model, where C is the maximum number of transfers allowed in a round.*

Proof. Let us denote the number of rounds required in an optimal solution for the full matching model and bounded-size matching model as OPT and OPT', respectively. Denote the number of rounds in our algorithm as t and t'.

Note that since we move data only to disks that need the data, the total number of data transfers performed by the algorithm is the minimum possible. Thus $OPT' \geq \sum_i |E_i|/C$. Since $t \leq \rho OPT$ and $OPT \leq OPT'$, we have $t \leq \rho OPT'$.

Therefore,

$$\begin{aligned} t' &= \sum_{i=1}^{t} \left\lceil \frac{|E_i|}{C} \right\rceil \\ &\leq \sum_{i=1}^{t} \left(\frac{|E_i| - 1}{C} + 1 \right) \\ &= \frac{1}{C} \sum_{i=1}^{t} |E_i| + t \left(1 - \frac{1}{C} \right) \\ &\leq OPT' + \rho OPT' \left(1 - \frac{1}{C} \right) \\ &= \left(1 + \rho \left(1 - \frac{1}{C} \right) \right) OPT'. \end{aligned}$$

∎

COROLLARY 4.2. *We have a* $1 + 9.5(1 - 1/C)$*-approximation algorithm for the bounded-size matching model.*

When we consider only move operations, we can obtain better bounds for the bounded-size matching model. Without space constraints, the problem can be reduced to edge-coloring multigraphs, which has a 1.1-approximation algorithm with a 0.8 additive term [22].

COROLLARY 4.3. *With only move operations, we have a* $1 + 1.1(1 - 1/C)$*-approximation algorithm with a* 0.8 *additive term for the bounded-size matching model.*

With space constraints, the algorithm by Hall et al. [11] gives $\frac{3}{\Delta_G}\lceil\frac{\Delta_G}{2}\rceil$-approximation. We thus obtain the following.

COROLLARY 4.4. *With only move operations and space constraints, we have a* $1 + \frac{3}{\Delta_G}\lceil\frac{\Delta_G}{2}\rceil(1 - \frac{1}{C})$*-approximation algorithm for the bounded-size matching model.*

In addition, using at most $n/3$ bypass nodes, Hall et al. [11] obtained algorithms which give $\frac{2}{\Delta_G}\lceil\frac{\Delta_G}{2}\rceil$-approximation without space constraints and $\frac{4}{\Delta_G}\lceil\frac{\Delta_G}{4}\rceil$-approximation with space constraints. The algorithms add one transfer for every odd cycle. For a transfer graph with sets of cycles of length 3, the algorithms take 4 rounds, while OPT' may take only 3 rounds. Thus we have $4OPT'/3 \geq \sum |E_i|/C$.

THEOREM 4.5. *When we allow only move operations and use at most* $n/3$ *bypass nodes, there is a* $\frac{4}{3} + \frac{2}{\Delta_G}\lceil\frac{\Delta_G}{2}\rceil(1 - \frac{1}{C})$*-approximation algorithm without space constraints and a* $\frac{4}{3} + \frac{4}{\Delta_G}\lceil\frac{\Delta_G}{4}\rceil(1 - \frac{1}{C})$*-approximation algorithm with space constraints.*

Proof. We use the same notation as in Theorem 4.1. Since $4OPT'/3 \geq \sum |E_i|/C$, we have

$$\begin{aligned} t' &= \sum_{i=1}^{t} \left\lceil \frac{|E_i|}{C} \right\rceil \\ &\leq \sum_{i=1}^{t} \left(\frac{|E_i| - 1}{C} + 1 \right) \\ &= \frac{1}{C} \sum_{i=1}^{t} |E_i| + t \left(1 - \frac{1}{C} \right) \\ &\leq \frac{4}{3} OPT' + \rho OPT' \left(1 - \frac{1}{C} \right) \\ &= \left(\frac{4}{3} + \rho \left(1 - \frac{1}{C} \right) \right) OPT'. \end{aligned}$$

□

Acknowledgments. We would like to thank Sudarshan Chawathe, Leana Golubchik, Liviu Iftode, and John Kubiatowicz for useful discussions related to the definition of the problem considered.

REFERENCES

[1] E. ANDERSON, J. HALL, J. HARTLINE, M. HOBBES, A. KARLIN, J. SAIA, R. SWAMINATHAN, AND J. WILKES, *An experimental study of data migration algorithms*, in Proceedings of the Workshop on Algorithm Engineering, Lecture Notes in Comput. Sci. 2141, Springer-Verlag, New York, 2001, pp. 145–158.

[2] B. BAKER AND R. SHOSTAK, *Gossips and telephones*, Discrete Math., 2 (1972), pp. 191–193.

[3] J. BERMOND, L. GARGANO, AND S. PERENNES, *Optimal sequential gossiping by short messages*, Discrete Appl. Math., 86 (1998), pp. 145–155.

[4] J. BERMOND, L. GARGANO, A. A. RESCIGNO, AND U. VACCARO, *Fast gossiping by short messages*, in Proceedings of the International Colloquium on Automata, Languages and Pro-

gramming, Lecture Notes in Comput. Sci. 944, Springer-Verlag, New York, 1995, pp. 135–146.
[5] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, American Elsevier, New York, 1977.
[6] R. T. Bumby, *A problem with telephones*, SIAM J. Algebraic Discrete Methods, 2 (1981), pp. 13–18.
[7] E. J. Cockayne and A. G. Thomason, *Optimal multimessage broadcasting in complete graphs*, Utilitas Math., 18 (1980), pp. 181–199.
[8] A. M. Farley, *Broadcast time in communication networks*, SIAM J. Appl. Math., 39 (1980), pp. 385–390.
[9] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu, *Approximation algorithms for data placement on parallel disks*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 2000), ACM, New York, SIAM, Philadelphia, 2000, pp. 223–232.
[10] L. Golubchik, S. Khuller, Y. A. Kim, S. Shargorodskaya, and Y. C. Wan, *Data Migration on Parallel Disks*, Technical Report CS-TR-4547, University of Maryland, 2003.
[11] J. Hall, J. Hartline, A. R. Karlin, J. Saia, and J. Wilkes, *On algorithms for efficient data migration*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (Washington, DC, 2001), ACM, New York, SIAM, Philadelphia, 2001, pp. 620–629.
[12] A. Hajnal, E. C. Milner, and E. Szemeredi, *A cure for the telephone disease*, Canad. Math. Bull., 15 (1972), pp. 447–450.
[13] S. M. Hedetniemi, S. T. Hedetniemi, and A. Liestman, *A survey of gossiping and broadcasting in communication networks*, Networks, 18 (1988), pp. 129–134.
[14] I. Holyer, *The NP-completeness of edge-coloring*, SIAM J. Comput., 10 (1981), pp. 718–720.
[15] J. Hromkovic, R. Klasing, B. Monien, and R. Peine, *Dissemination of information in interconnection networks (broadcasting and gossiping)*, in Combinatorial Network Theory, D.-Z. Du and D. F. Hsu, eds., Kluwer Academic, Dordrecht, The Netherlands, 1996, pp. 125–212.
[16] C. A. J. Hurkens, *Spreading gossip efficiently*, Nieuw Arch. Wiskd. (5), 1 (2000), pp. 208–210.
[17] S. Kashyap and S. Khuller, *Algorithms for non-uniform size data placement on parallel disks*, in Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science, Mumbai, India, Lecture Notes in Comput. Sci. 2914, Springer-Verlag, New York, 2003, pp. 265–276.
[18] S. Khuller, Y. A. Kim, and Y. C. Wan, *On generalized gossiping and broadcasting*, in Proceedings of the 11th Annual European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 2832, Springer-Verlag, New York, 2003, pp. 373–384.
[19] W. Knodel, *New gossips and telephones*, Discrete Math., 13 (1975), p. 95.
[20] H. M. Lee and G. J. Chang, *Set to set broadcasting in communication networks*, Discrete Appl. Math., 40 (1992), pp. 411–421.
[21] D. Liben-Nowell, *Gossip is synteny: Incomplete gossip and an exact algorithm for syntenic distance*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (Washington, DC, 2001), ACM, New York, SIAM, Philadelphia, 2001, pp. 177–185.
[22] T. Nishizeki and K. Kashiwagi, *On the* 1.1 *edge-coloring of multigraphs*, SIAM J. Discrete Math., 3 (1990), pp. 391–410.
[23] D. Richards and A. L. Liestman, *Generalizations of broadcasting and gossiping*, Networks, 18 (1988), pp. 125–138.
[24] H. Shachnai and T. Tamir, *On two class-constrained versions of the multiple knapsack problem*, Algorithmica, 29 (2001), pp. 442–467.
[25] H. Shachnai and T. Tamir, *Polynomial time approximation schemes for class-constrained packing problems*, in Approximation Algorithms for Combinatorial Optimization (Saarbrücken, 2000), Lecture Notes in Comput. Sci. 1913, Springer-Verlag, Berlin, 2000, pp. 238–249.
[26] C. E. Shannon, *A theorem on colouring lines of a network*, J. Math. Phys., 28 (1949), pp. 148–151.
[27] D. B. Shmoys and E. Tardos, *An approximation algorithm for the generalized assignment problem*, Math. Programming, A 62 (1993), pp. 461–474.
[28] R. Tijdeman, *On a telephone problem*, Nieuw Arch. Wisk. (3), 19 (1971), pp. 188–192.
[29] V. G. Vizing, *On an estimate of the chromatic class of a p-graph*, Diskret. Analiz., 3 (1964), pp. 25–30 (in Russian).

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 462–486

AN OPTIMAL COMPETITIVE STRATEGY FOR WALKING IN STREETS*

CHRISTIAN ICKING†, ROLF KLEIN‡, ELMAR LANGETEPE‡, SVEN SCHUIERER§, AND INES SEMRAU¶

Abstract. A simple polygon P with two distinguished vertices, s and t, is called a *street* if the two boundary chains from s to t are mutually weakly visible. We present an on-line strategy that walks from s to t, in any unknown street, on a path at most $\sqrt{2}$ times longer than the shortest path. This matches the best lower bound previously known and settles an open problem in the area of competitive path planning. (The result was simultaneously and independently obtained by the first three authors and by the last two authors. Both papers, [C. Icking, R. Klein, and E. Langetepe, *Proceedings of the* 16*th Symposium on Theoretical Aspects in Computer Science*, Lecture Notes in Comput. Sci. 1563, Springer-Verlag, Berlin, 1999, pp. 110–120] and [S. Schuierer and I. Semrau, *Proceedings of the* 16*th Symposium on Theoretical Aspects of Computer Science*, pp. 121–131], were presented at STACS'99. The present paper contains a joint full version.)

Key words. computational geometry, autonomous robot, competitive strategy, LR-visibility, on-line navigation, path planning, polygon, street

AMS subject classifications. 68Q17, 68Q25, 68R01, 68W40

DOI. 10.1137/S0097539702419352

1. Introduction.

The path planning problem of autonomous mobile robots has received a lot of attention in the communities of robotics, computational geometry, and on-line algorithms; see, e.g., Rao et al. [37], Blum, Raghavan, and Schieber [6], and the surveys by Berman [4] in Fiat and Woeginger [15] and by Mitchell [35] in Sack and Urrutia [38].

In on-line navigation one has to perform a certain task in an initially unknown environment. We are interested in strategies that are correct, in that the objective will always be achieved whenever this is possible, and in performance guarantees of the following kind. Given a navigation problem Q, we want to relate the cost of solving any problem instance $P \in Q$ by means of strategy S to the cost of solving P optimally, using an off-line strategy. If the former never exceeds the latter times a certain constant factor, c, then strategy S is said to be a *c-competitive* solution of Q; this notion was coined by Sleator and Tarjan in their seminal paper [43]. Surveys on general on-line algorithms can be found in Fiat and Woeginger [15] and Ottmann, Schuierer, and Hipke [36].

Given an on-line problem Q, three questions arise: Does a competitive solution exist? If S is a solution, what is its true *competitive factor*, i.e., the smallest c such that S is c-competitive? And finally, what is the smallest factor c that can be attained by *any* strategy solving Q? This number is called the *competitive complexity* of problem Q.

There are not so many navigation tasks we are aware of whose competitive com-

*Received by the editors December 3, 2002; accepted for publication (in revised form) August 4, 2003; published electronically February 24, 2004. This research was partially supported by the Deutsche Forschungsgemeinschaft, grant Kl 655/8-3.
http://www.siam.org/journals/sicomp/33-2/41935.html
†FernUniversität Hagen, Praktische Informatik VI, D-58084 Hagen, Germany.
‡Universität Bonn, Institut für Informatik I, D-53117 Bonn, Germany.
§Novartis Institutes for Biomedical Research, Lichtstr. 35, CH-4002 Basel, Switzerland.
¶Universität Freiburg, Institut für Informatik, Am Flughafen 17, D-79110 Freiburg, Germany.

plexities are precisely known. One of them is searching for a point on m halflines that meet at the start point; see Baeza-Yates, Culberson, and Rawling [2] and also Bellman [3], Gal [16], Schuierer [39], López-Ortiz and Schuierer [34], and Alpern and Gal [1].

Another one is looking around a corner in a wall; see Icking, Klein, and Ma [22]. Recently, matching lower and upper bounds for some restricted on-line TSP problems where shown by Blom et al. [5]. More often, there is a gap between the smallest competitive factor known and the best lower bound, such as in the polygon exploration problem; see Hoffmann et al. [18].

In this paper we prove that walking in an unknown street is of competitive complexity $\sqrt{2}$, thus settling a problem that has been open for a decade. A *street* is a simple polygon P with two vertices, s and t, that mark the start and target point of the walk, subject to the condition that the two boundary chains connecting s to t are mutually weakly visible[1]; see Figure 1.1 for an example. This is equivalent to saying that from each s-to-t path inside P, each point of P can be seen at least once. Streets were introduced in Klein [24] to model racetracks and rivers like the Rhine that may contain curves and bays but no cul-de-sacs winding away from the main route. It was shown in [24, 25] that there exists a strategy that is competitive with a factor of 5.72, and that no factor smaller than $\sqrt{2}$ can be achieved, not even by a randomized strategy.

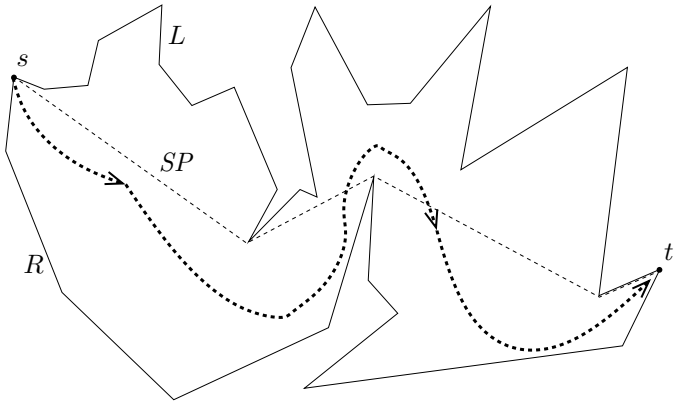


FIG. 1.1. *A street.*

Since then, the street problem has attracted considerable attention. Some research was devoted to structural properties. Tseng, Heffernan, and Lee [44] have shown how to report all pairs of vertices (s, t) of a given polygon for which it is a street; for star-shaped polygons many such vertex pairs exist. Das, Heffernan, and Narasimhan [10] have improved on this result by giving an optimal linear time algorithm. Ghosh and Saluja [17] have described how to walk an unknown street with a minimum number of turns.

For arbitrary polygons it is quite easy to see that in general no strategy can guarantee a search path whose length is at most a constant times the length of the shortest path from start to target.[2] Therefore, some researchers have designed competitive search strategies for classes of polygons more general than streets; see Datta

[1] Two sets are mutually weakly visible if each point of one set can see at least one point of the other.

[2] If n L-shaped legs of unit length lead away from a central place, the search path can be of length $2n - 1$, in the worst case, while the shortest path is of length 1.

and Icking [13, 14], Datta, Hipke, and Schuierer [12], and López-Ortiz and Schuierer [30, 33].

Other authors have shown that more general search problems can be solved for the original street polygons. Indeed, not only can we walk to vertex t starting from vertex s, where s, t are the two special vertices defining the street, but it has also been shown in Bröcker and Schuierer [8] and Bröcker and López-Ortiz [7] that one can find any boundary point from any starting point on the boundary of a street by means of a 69.216-competitive strategy.

Carlsson and Nilsson [9] have shown that the art gallery problem remains NP-hard for street polygons. However, the problem of computing the minimum number of guards located on a given watchman route can be solved very efficiently for streets, while it is NP-hard for general polygons.

Other research has addressed the gap between the $\sqrt{2}$ lower bound and the first upper bound of 5.72 for the original street problem. The upper bound was lowered to 4.44 in Icking [19], then to 2.61 in Kleinberg [26], to 2.05 in López-Ortiz and Schuierer [29], and to 1.73 in López-Ortiz and Schuierer [31]. López-Ortiz and Schuierer [32] showed that a particular strategy called CAB has the true competitive ratio 1.6837. Using different search strategies, the upper bound was further decreased to 1.57 in Semrau [42], and to 1.51 in Icking et al. [23]. Further attempts were made by Dasgupta, Chakrabarti, and DeSarkar [11] and by Kranakis and Spatharis [27].

But it remained an open question whether there existed a search strategy with optimal competitive factor $\sqrt{2}$; this was mentioned open problem no. 13 in Mitchell [35].

In this paper the problem is finally solved. We introduce a new strategy and prove that the search path it generates, in any particular street, is at most $\sqrt{2}$ times the length of the shortest path from s to t. Unlike many approaches discussed in previous work, the optimal strategy we are presenting here is not a mere artifact. Rather, its definition is well motivated by backward reasoning, as we shall now explain.

The crucial subproblem can be parametrized by a single angle, ϕ. For each possible value of ϕ a lower bound can be established; see section 3.1. For the maximum value $\phi = \pi$ the existence of a strategy matching this bound follows from the properties of a street. We state a requirement in section 3.2 that would allow us to extend an optimal strategy from a given value of ϕ to smaller values. From this requirement we can infer how the strategy should proceed; see section 3.3. One of the problems is to prove that the requirement can be fulfilled; see section 3.4.

2. Definitions and basic properties. First, we briefly recall some basic definitions.

A simple polygon P is considered as a room, with its edges as opaque walls. By ∂P we denote the boundary of polygon P. Two points inside P are mutually *visible*, i.e., see each other, if the connecting line segment is contained within P. As usual, two sets of points are said to be *mutually weakly visible* if each point of one set can see at least one point of the other set.

DEFINITION 2.1. *A simple polygon P in the plane with two distinguished vertices s and t is called a* street *if the two boundary chains from s to t are weakly mutually visible; for an example, see Figure* 1.1. *Streets are sometimes also called* LR-visible polygons [10, 44]*, where L and R denote the left and the right boundary chains from s to t, respectively.*

A strategy for searching a target in an unknown street is an on-line algorithm for a mobile robot, modeled by a point, that starts at vertex s, moves around inside the

polygon, and eventually arrives at the target, t. The robot is equipped with a vision system that provides the visibility polygon for its actual position at each time, and everything that has been visible is memorized. When the target becomes visible the robot goes there, and its task is accomplished.

As the room's floor plan is not known in advance, the robot's path can be longer than the shortest path, SP, from s to t inside P. Our goal is to bound that detour.

DEFINITION 2.2. *A strategy for searching a target, t, in a street is called* competitive with factor c *(or c-*competitive*, for short) if its path is never longer than c times the length of the shortest path from s to t.*

All existing algorithms for walking along a street are making use of some geometric properties that can be derived from the definition of a street; these facts and their complete proofs can be found in [25]. For the convenience of the reader these properties, together with an outline of their proofs, will now be stated.

First, we consider the situation at the beginning. As the robot starts from vertex s, it may not be able to see the whole polygon. The parts invisible to the robot are called *caves*. Each cave is hidden behind a reflex vertex, v, which is one whose internal angle exceeds 180°. Such a reflex vertex—and its associated cave—is called *left* if its adjacent edges in ∂P lie to the left of the ray emanating from the robot's position through v. Right reflex vertices are defined analogously.

If s is the start point of a street, these caves can occur only in a certain order. As the robot scans ∂P in a clockwise direction, it encounters a consecutive sequence of left caves with left reflex vertices $v_l^{-a}, v_l^{-a+1}, \ldots, v_l^0, v_l^1$ followed by a consecutive sequence of right caves with right reflex vertices $v_r^1, v_r^0, \ldots, v_r^{-b+1}, v_r^{-b}$; see Figure 2.1. Either sequence can be empty. The reason for this ordering is that a right cave cannot be predecessor of a left cave. Assuming the contrary, let v be a right reflex vertex that appears before the left reflex vertex w in clockwise order on ∂P. Let v^- be the predecessor vertex of v, and let w^+ denote the successor of w. If v were a vertex of chain L, then v^- would not be able to see a point of chain R, in contradiction to the street property. Thus, $v \in R$ holds. Similarly, we have $w \in L$. But this is impossible since L and R are connected and meet at s.

Of all the left caves visible from s only the clockwise-most can contain the target; see Figure 2.1. The reason is similar to the proof above. In fact, the reflex vertex v_l of the clockwise-most left cave cannot belong to chain R, or its successor on the boundary would not be able to see a point of L. Analogously, only the counterclockwise-most right cave with reflex vertex v_r can contain the target vertex t.

Consequently, if only v_l exists, then its cave must contain the target, and the robot walks straight to v_l; see Figure 2.1(ii). We observe that this reflex vertex must also be visited by the shortest path from s to t. The same holds if only v_r exists.

A more interesting situation arises if both v_l and v_r exist. Then the target can be situated in either of their caves, but the robot does not know in which one; see Figure 2.1(i). We call this a *funnel situation*. The angle, ϕ, between the directions from the actual position to v_l and to v_r is called the *opening angle*; it is always smaller than π, as another consequence of the street property. Most search strategies cause the robot to walk into this funnel of angle ϕ. They differ in choosing the direction into the funnel.

As the robot leaves the start vertex s, the vertices v_l and v_r are maintained by the robot. Essentially, v_l is the reflex vertex of the clockwise-most left cave in front of the robot, and v_r is the entrance vertex of the counterclockwise-most right cave. The vertices v_l and v_r are known to belong to L and R, respectively; but the horizon,

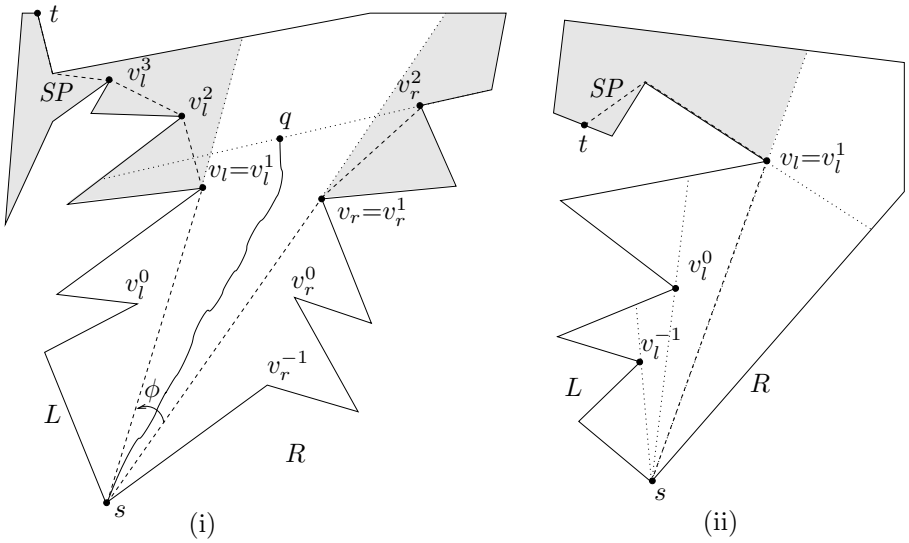


FIG. 2.1. *Typical situations in streets.*

that is, the boundary part between v_l and v_r, can belong to either chain, depending on the position of the target.

To summarize, the robot behaves as follows. If the target is visible from the robot's current position, the robot walks straight to the target. If only one of the vertices v_l, v_r exists, then the robot walks straight to this vertex, which is also visited by the shortest path. If both v_l and v_r exist, then the robot walks into the funnel defined by its current position, c_0, and by v_l and v_r.

As the robot moves into the funnel, three events can happen. The most important event occurs when a new reflex vertex, say $v_l{}'$, appears behind vertex v_l. In this case, we know from the discussion above that the target cannot be contained in the cave of $v_l{}'$; it must be situated in the caves of $v_l{}'$ or v_r. Now the robot proceeds with $v_l{}'$ and v_r. This event can occur repeatedly on both sides. It generates convex chains of reflex vertices $v_l^1, v_l^2, \ldots, v_l^m$ and $v_r^1, v_r^2, \ldots, v_r^n$ that form a funnel with apex c_0.

Another event can occur when one of the two innermost caves, say the left, becomes completely visible. In this case, the target must be situated in the right cave, and the robot walks to its associated reflex vertex, v_r^n. On reaching v_r^n, the funnel situation is resolved, and we know that the chain $v_r^1, v_r^2, \ldots, v_r^n$ belongs to the shortest path from s to t.

The third event occurs when the target becomes visible, e.g., in the right cave; then the robot walks straight to t, visiting the reflex vertex v_r^n on the way.

This analysis shows that detour is only caused by funnels, and that the overall competitive factor of a search strategy for streets depends only on its performance in funnels.

As a consequence, we can restrict our attention to the subclass of funnel polygons. They consist of two chains of reflex vertices with a common start point s; see Figure 2.2 for an example. The two reflex chains end in vertices t_l and t_r, respectively, and the line segment $t_l\, t_r$ closes the polygon. A funnel polygon represents a funnel situation in which the target t lies arbitrarily close behind either t_l or t_r, and the strategy will know which case applies only when the line segment $t_l t_r$ is reached. For analyzing

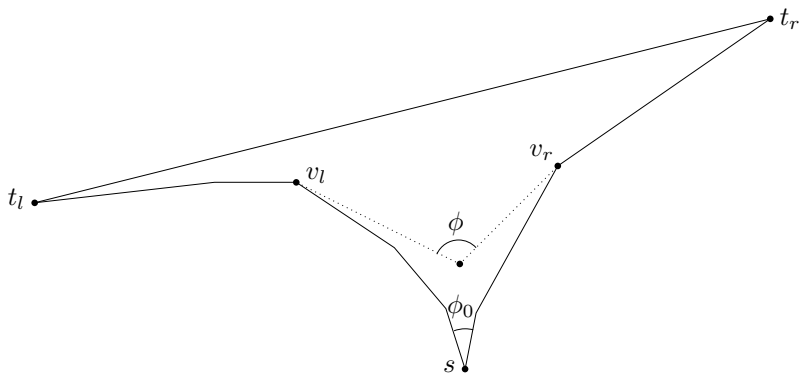


FIG. 2.2. *A funnel.*

a strategy, both cases have to be considered and the worse of them determines the competitive factor. Other funnel situations, which end before line segment $t_l t_r$ is reached or where the goal is further away from t_l or t_r, will produce a smaller detour.

Since the walking direction is always within the opening angle, ϕ is always *strictly increasing*. It starts at the angle, ϕ_0, between the two edges adjacent to s, and reaches, but never exceeds, 180° when finally the goal becomes visible. By this property, it is quite natural to take the opening angle ϕ for *parameterizing a strategy*.

3. A strategy which always takes the worst case into account. In the previous section we have seen that a crucial situation occurs when the robot is faced with two caves, one left and one right, and does not know in which of them the target, t, is situated. This situation can be parametrized by the funnel's opening angle ϕ.

Let us assume that $\frac{\pi}{2} \leq \phi$ holds. We will see in section 3.1 that for each value of ϕ, a *lower* bound for the competitive ratio is given by

$$K_\phi = \sqrt{1 + \sin \phi}.$$

If $\phi = \pi$, then $K_\pi = 1$. The street properties ensure that the robot is able to look into the caves and see the target. Hence, the optimal strategy is given by walking straight to t. That means, for $\phi = \pi$ we have a strategy matching the lower bound.

Now assume that $\frac{\pi}{2} \leq \phi_1 < \phi_2 < \pi$ holds, and that we have already found a strategy with (optimum) competitive factor K_{ϕ_2} for all opening angles $\geq \phi_2$. We would like to extend it to a K_{ϕ_1}-competitive strategy for opening angles ϕ_1. This is possible iff a certain geometric condition can be met, which will be stated in section 3.2. This condition gives rise to a certain curve (see section 3.3), and this curve will then be shown to have the required properties in section 3.4. Finally, in section 3.5, we deal with opening angles less than π.

3.1. A generalized lower bound. We start with a generalized lower bound for initial opening angles $\geq \frac{\pi}{2}$. For an arbitrary angle ϕ, let

$$K_\phi := \sqrt{1 + \sin \phi}\,.$$

LEMMA 3.1. *Assume an initial opening angle $\phi_0 \geq \frac{\pi}{2}$. Then no strategy can guarantee a smaller competitive factor than K_{ϕ_0}.*

Proof. We take an isosceles triangle with an angle ϕ_0 at vertex s; the other vertices are t_l and t_r; see Figure 3.1.

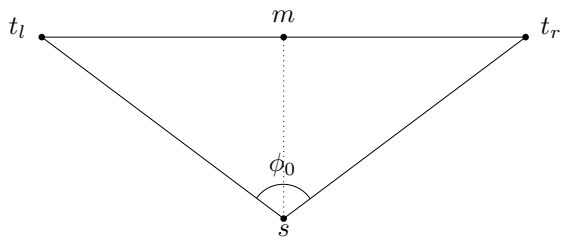


FIG. 3.1. *Establishing a generalized lower bound.*

The goal becomes visible only when the line segment $t_l\, t_r$ is reached. If this happens to the left of the midpoint m, the goal may be to the right, and vice versa. In any case the path length is at least the distance from s to m plus the distance from m to t_l. For the ratio, c, of the path length to the shortest path we obtain by simple trigonometry

$$c \geq \cos\frac{\phi_0}{2} + \sin\frac{\phi_0}{2} = \sqrt{1+\sin\phi_0} = K_{\phi_0}\,. \qquad \square$$

For $\phi_0 = \frac{\pi}{2}$, we have the well-known lower bound of $\sqrt{2}$ stemming from a rectangular isosceles triangle; see Klein [25].

Note that the lower bound K_{ϕ_0} also applies to any nonsymmetric situation, since at the start the funnel is unknown except for the two edges adjacent to s, and it may turn into a nearly symmetric case immediately after the start. In other words this means that for an initial opening angle ϕ_0, a competitive factor of K_{ϕ_0} is always the best we can hope for.

In the following we develop a strategy that is K_ϕ-competitive in all funnel polygons of opening angle ϕ.

3.2. Sufficient requirements for an optimal strategy. In a funnel with opening angle π the goal is visible and there is a trivial strategy that achieves the optimal competitive factor $K_\pi = 1$. So we look backwards to decreasing angles.

Let us assume for the moment that the funnel is a triangle, and that we have a strategy with a competitive factor of K_{ϕ_2} for all triangular funnels with initial opening angle ϕ_2. How can we extend this to initial opening angles ϕ_1 with $\pi \geq \phi_2 > \phi_1 \geq \frac{\pi}{2}$?

Starting with an angle ϕ_1 at point p_1 we walk a certain path of length w until we reach an angle of ϕ_2 at point p_2, from where we can continue with the known strategy; see Figure 3.2. We assume for the moment that the left and right reflex vertices, v_l and v_r as defined in section 2, do not change.

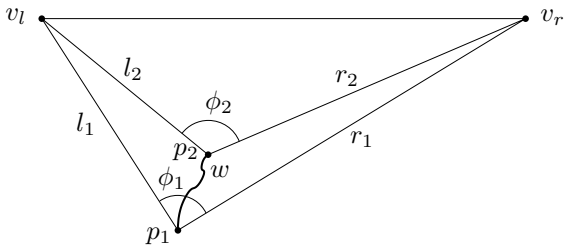


FIG. 3.2. *Getting from angle ϕ_1 to ϕ_2.*

Let l_1 and l_2 denote the distances from p_1, respectively, p_2, to v_l at the left side, and r_1 and r_2 the corresponding distances at the right. If $t = v_l$, the length of the robot's path from p_1 to t is not greater than $w + K_{\phi_2} l_2$. If now $K_{\phi_1} l_1 \geq w + K_{\phi_2} l_2$ holds and the analogous inequality $K_{\phi_1} r_1 \geq w + K_{\phi_2} r_2$ for the right side, we have a competitive factor not bigger than K_{ϕ_1} for triangles with initial opening angle ϕ_1. By combining the two inequalities we can express the condition as

$$w \leq \min(K_{\phi_1} l_1 - K_{\phi_2} l_2, K_{\phi_1} r_1 - K_{\phi_2} r_2)\,, \tag{3.1}$$

which will prove useful later on.

Note that condition (3.1) is additive in the following sense. If it holds for a path w_{12} from ϕ_1 to ϕ_2 and for a continuing path w_{23} from ϕ_2 to ϕ_3, it is also true for the combined path $w_{12} + w_{23}$ from ϕ_1 to ϕ_3. This will turn out to be very useful: if (3.1) holds for arbitrarily small, successive steps w, then it is also true for all bigger ones.

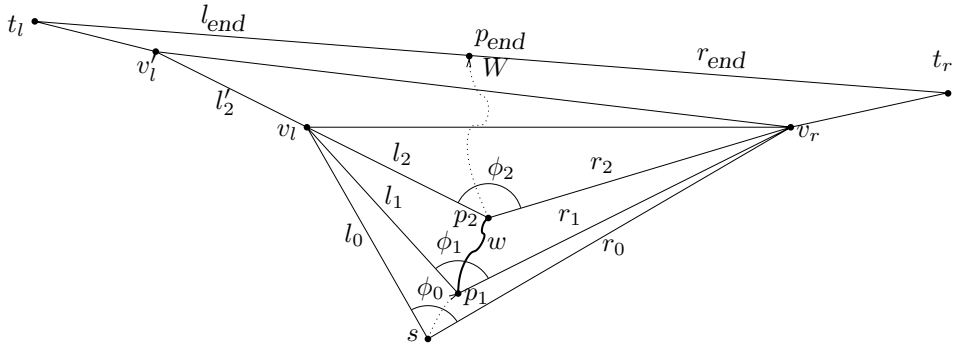


FIG. 3.3. *When p_2 is reached, the most advanced visible point to the left jumps from v_l to v_l'.*

Now let us go further backwards and observe what happens if one of the current vertices v_l or v_r change. We assume that condition (3.1) holds for path w from p_1 to p_2 and that v_l changes at p_2; see Figure 3.3. The visible left chain is extended by l_2'. Nothing changes on the right side of the funnel, and for the left side of the funnel we have

$$\begin{aligned} w \leq K_{\phi_1} l_1 - K_{\phi_2} l_2 &= K_{\phi_1} l_1 - K_{\phi_2} l_2 + K_{\phi_2} l_2' - K_{\phi_2} l_2' \\ &\leq K_{\phi_1}(l_1 + l_2') - K_{\phi_2}(l_2 + l_2')\,. \end{aligned} \tag{3.2}$$

The last inequality holds because $K_\phi = \sqrt{1 + \sin\phi}$ is decreasing with increasing $\phi \in [\frac{\pi}{2}, \pi]$. Here, $l_1 + l_2'$ and $l_2 + l_2'$ are the lengths of the shortest paths from p_1 and p_2 to v_l', respectively. But (3.2) in fact means that (3.1) remains valid even if changes of v_l or v_r occur.

Under the assumption that (3.1) holds for all small steps where v_l and v_r do not change we can make use of the additivity of (3.1) and obtain the following expression for the path length, W, from an initial opening angle ϕ_0 to the point p_{end} where the line segment $t_l\, t_r$ is reached; see Figure 3.3.

$$W \leq \min\Big(\begin{array}{l} K_{\phi_0}(\text{length of left chain}) - K_\pi l_{end}, \\ K_{\phi_0}(\text{length of right chain}) - K_\pi r_{end} \end{array}\Big).$$

But, since $K_\pi = 1$, this inequality exactly means that we have a competitive factor not larger than K_{ϕ_0}. Only a curve remains to be found that satisfies (3.1) for small steps.

3.3. Developing the curve. One could try to satisfy condition (3.1) by analyzing, for fixed p_1, ϕ_1, and ϕ_2, which points p_2 meet that requirement. To avoid this tedious task, we argue as follows. For fixed ϕ_2, the point p_2 lies on a circular arc U_{ϕ_2} through v_l and v_r. While p_2 moves along the arc U_{ϕ_2}, the length l_2 is strictly increasing while r_2 is strictly decreasing. Heuristically, we maximize our chances to satisfy (3.1) if we require that

$$K_{\phi_1} l_1 - K_{\phi_2} l_2 = K_{\phi_1} r_1 - K_{\phi_2} r_2$$

or, equivalently,

$$K_{\phi_2}(l_2 - r_2) = K_{\phi_1}(l_1 - r_1)\,. \tag{3.3}$$

We claim that inside the triangle defined by ϕ_1 and segments of length l_1 and r_1 there exists a point p_2 on U_{ϕ_2} that satisfies (3.3). Indeed, while p_2 moves along U_{ϕ_2} between the intersections of U_{ϕ_2} with the segments of length l_1 and r_1, the continuous expression $K_{\phi_2}(l_2 - r_2) - K_{\phi_1}(l_1 - r_1)$ changes its sign; see Figure 3.4. If p_2 is the intersection of U_{ϕ_2} with the segment of length r_1, we have $r_2 < r_1$ and $K_{\phi_2} < K_{\phi_1}$, and $K_{\phi_2}(l_2 - r_2) - K_{\phi_1}(l_1 - r_1)$ is positive if $K_{\phi_1} l_1 \le K_{\phi_2} l_2$. Using the law of sine, $K_{\phi_1} l_1 \le K_{\phi_2} l_2$ is equivalent to $\frac{K_{\phi_1}}{\sin\phi_1} \le \frac{K_{\phi_2}}{\sin\phi_2}$. The expression $\frac{K_\phi}{\sin\phi}$ is monotonically increasing for $\frac{\pi}{2} \le \phi < \pi$. For the same reason $K_{\phi_2}(l_2 - r_2) - K_{\phi_1}(l_1 - r_1) < 0$ holds if p_2 is the intersection of U_{ϕ_2} with the segment of length l_1.

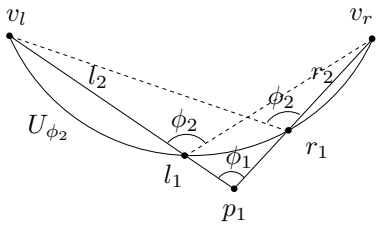


FIG. 3.4. *$K_{\phi_2}(l_2 - r_2) - K_{\phi_1}(l_1 - r_1)$ changes its sign along the circular arc U_{ϕ_2}.*

Altogether, if we start with the initial values ϕ_0, l_0, r_0, we define the fixed constant $A := K_{\phi_0}(l_0 - r_0)$, and for any $\phi_0 \le \phi \le \pi$ with corresponding lengths l_ϕ and r_ϕ we want that

$$K_\phi(l_\phi - r_\phi) = A \tag{3.4}$$

holds as long as v_l and v_r do not change. In the symmetric case $l_0 = r_0$ this condition means that we walk along the bisector of v_l and v_r. Otherwise, condition (3.4) defines a curve which can be determined in the following way. We choose a coordinate system with horizontal axis $v_l\, v_r$, the midpoint being the origin. We scale the coordinate system so that the distance from v_l to v_r equals 1. With this choice we have

$$|A| = |K_\phi(l_\phi - r_\phi)| \le K_\phi = \sqrt{1 + \sin\phi} \tag{3.5}$$

for every l_ϕ, r_ϕ, and K_ϕ in the triangle defined by ϕ_0, l_0, r_0.

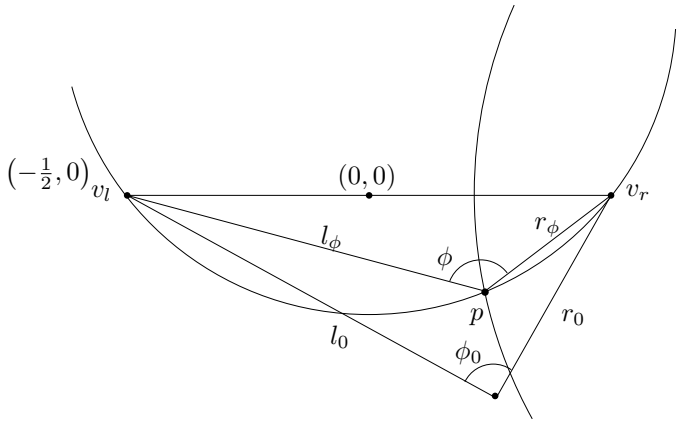


FIG. 3.5. *The right arc of the hyperbola defined by v_l, v_r, and $(l-r) = \frac{A}{K_\phi}$ and the circle through v_l and v_r defined by angle ϕ meet in $p = (X(\phi), Y(\phi))$.*

W.l.o.g. let $l_0 > r_0$. For any $\phi_0 \le \phi < \pi$ the corresponding point of the curve is the intersection of the hyperbola

$$\frac{X^2}{\left(\frac{A}{2K_\phi}\right)^2} - \frac{Y^2}{\left(\frac{1}{2}\right)^2 - \left(\frac{A}{2K_\phi}\right)^2} = 1 \tag{3.6}$$

with the circle

$$X^2 + \left(Y + \frac{\cot\phi}{2}\right)^2 = \frac{1}{4\sin^2\phi}; \tag{3.7}$$

see Figure 3.5 and the details found in section A.1 of the appendix.

Solving these equations leads us, after some transformations, to the following solutions (for details see section A.2 of the appendix):

$$X(\phi) = \frac{A}{2} \cdot \frac{\cot\frac{\phi}{2}}{1+\sin\phi}\sqrt{\left(1+\tan\frac{\phi}{2}\right)^2 - A^2}, \tag{3.8}$$

$$Y(\phi) = \frac{1}{2}\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi} - 1\right). \tag{3.9}$$

Since $|A| < \sqrt{1+\sin\phi} < 1 + \tan\frac{\phi}{2}$ holds, the functions $X(\phi)$ and $Y(\phi)$ are well defined and continuous while the curve stays below the line segment $v_l v_r$.

Figure 3.6 shows how these curves look for all possible values of ϕ and A. All points with an initial opening angle of $\frac{\pi}{2}$ lie on the lower half circle.

Two cases can be distinguished. If $|A| < 1$, then the curves can be continuously extended to a point $(\frac{A}{2}, 0)$ on the line segment $v_l\, v_r$. For $|A| > 1$ the curves end up in v_l and v_r, respectively, with a limit of opening angles $\phi = \pi - \arcsin(A^2 - 1)$, which satisfies $X(\phi) = \pm\frac{1}{2}$, $Y(\phi) = 0$, and $|A| = K_\phi$. The curves for the limiting cases $|A| = 1$ are emphasized with a thick line in Figure 3.6.

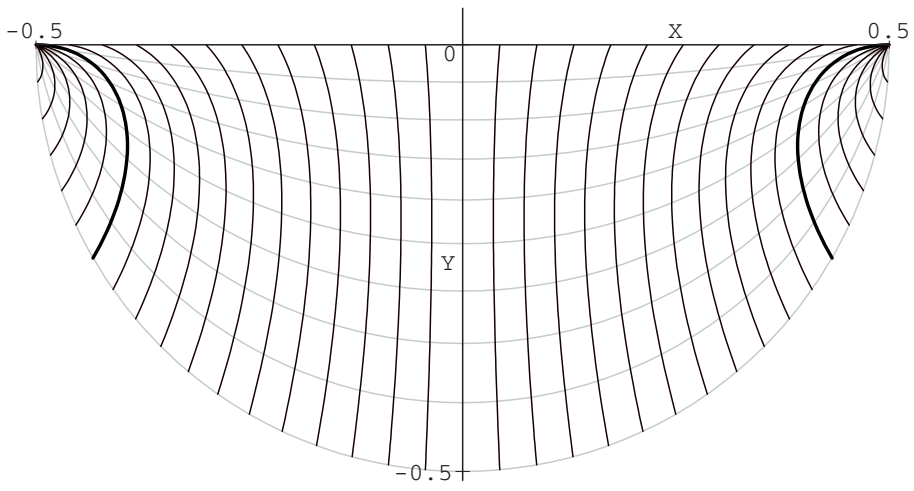


FIG. 3.6. *The curves fulfilling condition* (3.4) *for all values of* ϕ *and* A.

3.4. Checking the requirements. We want to check that the curve defined by (3.8) and (3.9) in section 3.3 satisfies condition (3.1). The arc length of the curve from angle ϕ_1 to ϕ_2 has to be compared to the right side of (3.1). Because of (3.3) the min in (3.1) can be dropped.

For $l_0 = r_0$ we trivially have equality in (3.1). W.l.o.g. we can assume $l_0 > r_0$ and $A > 0$. The other case is symmetric. It suffices to check

$$\int_{\phi_1}^{\phi_2} \sqrt{X'(\phi)^2 + Y'(\phi)^2}\, d\phi \le K_{\phi_1} l_{\phi_1} - K_{\phi_2} l_{\phi_2} \qquad \text{for all } \frac{\pi}{2} \le \phi_1 < \phi_2 < \pi\,.$$

Here, $X'(\phi)$ and $Y'(\phi)$ denote the derivatives of $X(\phi)$ and $Y(\phi)$ from (3.8) and (3.9) in ϕ. The inequality is equivalent to

$$\int_{\phi_1}^{\phi_2} \sqrt{X'(\phi)^2 + Y'(\phi)^2}\, d\phi \le \int_{\phi_1}^{\phi_2} -(K_\phi l_\phi)'\, d\phi \qquad \text{for all } \frac{\pi}{2} \le \phi_1 < \phi_2 < \pi\,, \tag{3.10}$$

since $K_\phi l_\phi$ is a differentiable function in ϕ. It is sufficient to show that in (3.10) one integrant dominates the other. In the following we will try to simplify this task.

By some transformations (for details see section A.3 of the appendix), we obtain

$$l_\phi = \frac{K_\phi}{A} X(\phi) + \frac{A}{2K_\phi} \quad \text{and therefore} \quad K_\phi l_\phi = \frac{K_\phi^2}{A} X(\phi) + \frac{A}{2}\,. \tag{3.11}$$

Thanks to an idea by Seidel [41] we can make use of the substitution $t = \tan\frac{\phi}{2}$

in K_ϕ, $X(\phi)$, $Y(\phi)$, and $K_\phi l_\phi$. Thus, we get the functions

$$\overline{K}(t) := \sqrt{\frac{(1+t)^2}{1+t^2}},$$
$$\overline{X}(t) := \frac{A}{2t\overline{K}^2(t)}\sqrt{(1+t)^2 - A^2} = \frac{A(1+t^2)\sqrt{(1+t)^2-A^2}}{2t(1+t)^2},$$
$$\overline{Y}(t) := \left(\frac{A^2}{\overline{K}^2(t)} - 1\right)\frac{1}{2t} = \frac{A^2(1+t^2)-(1+t)^2}{2t(1+t)^2},$$
$$\overline{Kl}(t) := \frac{\overline{K}^2(t)}{A}\overline{X}(t) + \frac{A}{2} = \frac{\sqrt{(1+t)^2-A^2}}{2t} + \frac{A}{2}$$

with the identities $\overline{K}(\tan\frac{\phi}{2}) = K_\phi$, $\overline{X}(\tan\frac{\phi}{2}) = X(\phi)$, $\overline{Y}(\tan\frac{\phi}{2}) = Y(\phi)$, and $\overline{Kl}(\tan\frac{\phi}{2}) = K_\phi l_\phi$. The simple identities are proven in section A.4 of the appendix.

We will now simplify (3.10) using terms of the identities above. The left-hand side of (3.10) is equivalent to

$$\begin{aligned}(3.12)\quad &\int_{\phi_1}^{\phi_2} \sqrt{\left(\frac{d}{d\phi}\left(\overline{X}\left(\tan(\phi/2)\right)\right)\right)^2 + \left(\frac{d}{d\phi}\left(\overline{Y}\left(\tan(\phi/2)\right)\right)\right)^2}\, d\phi \\ &= \int_{\phi_1}^{\phi_2} \frac{d}{d\phi}\left(\tan(\phi/2)\right)\sqrt{\left(\left(\frac{d}{dt}\overline{X}\right)\left(\tan(\phi/2)\right)\right)^2 + \left(\left(\frac{d}{dt}\overline{Y}\right)\left(\tan(\phi/2)\right)\right)^2}\, d\phi,\end{aligned}$$

whereas the right-hand side of (3.10) is equivalent to

$$(3.13)\quad \int_{\phi}^{\phi_2} -\frac{d}{d\phi}\left(\overline{Kl}\left(\tan(\phi/2)\right)\right)\, d\phi = \int_{\phi}^{\phi_2} -\frac{d}{d\phi}\left(\tan(\phi/2)\right)\left(\frac{d}{dt}\overline{Kl}\right)\left(\tan(\phi/2)\right)\, d\phi.$$

By applying the rule of substitution to (3.12) and (3.13), (3.10) is equivalent to

$$(3.14)\quad \int_{\tan\frac{\phi_1}{2}}^{\tan\frac{\phi_2}{2}} \sqrt{\left(\frac{d}{dt}\overline{X}(t)\right)^2 + \left(\frac{d}{dt}\overline{Y}(t)\right)^2}\, dt \le \int_{\tan\frac{\phi_1}{2}}^{\tan\frac{\phi_2}{2}} -\frac{d}{dt}\overline{Kl}(t)\, dt\,.$$

The function $\tan\frac{\phi}{2}$ is positive, continuous, and monotonically increasing for $\frac{\pi}{2} \le \phi < \pi$. Then it suffices to show that in (3.14) one integrand dominates the other one for every t in the integration interval $[\tan\frac{\phi_1}{2}, \tan\frac{\phi_2}{2}]$ for all $\frac{\pi}{2} \le \phi_1 < \phi_2 < \pi$. We make use of the following facts. For every $t \in [\tan\frac{\phi_1}{2}, \tan\frac{\phi_2}{2}]$ there is always a unique $\phi \in [\phi_1, \phi_2]$ with $t = \tan\frac{\phi}{2}$. Additionally we can assume $A < \sqrt{1+\sin\phi}$ from (3.5).

Altogether, it suffices to prove that

$$(3.15)\quad \sqrt{\overline{X}'(t)^2 + \overline{Y}'(t)^2} \le -\overline{Kl}'(t)$$

holds for $t = \tan\frac{\phi}{2}$, $\frac{\pi}{2} \le \phi < \pi$, and $0 < A < \sqrt{1+\sin\phi}$. Here for convenience $\overline{X}'(t)$, $\overline{Y}'(t)$, and $\overline{Kl}'(t)$ denote the derivatives of the corresponding functions in t. We insert the following identities (see section A.5 of the appendix for details) into (3.15).

$$-\overline{Kl}'(t) = \frac{(1+t) - A^2}{2t^2\sqrt{(1+t)^2 - A^2}},$$

$$\overline{X}'(t) = \frac{A\left(A^2\left((1+t)^3 - 4t^2\right) - 1 - 4t - 4t^2 + t^4\right)}{2(1+t)^3 t^2\sqrt{(1+t)^2 - A^2}},$$

$$\overline{Y}'(t) = \frac{(1+t)^3 - A^2\left((1+t)^3 - 4t^2\right)}{2(1+t)^3 t^2}\,.$$

Note that $-\overline{Kl}'(t) > 0$ follows from $t \geq \tan\frac{\pi}{4} = 1$ and $A^2 < 2$; see (3.5). Thus, after squaring, the following remains to be shown:

$$F(t,A) \geq 0 \text{ for all } t = \tan\frac{\phi}{2},\ \frac{\pi}{2} \leq \phi < \pi, \text{ and } 0 < A < \sqrt{1+\sin\phi} \text{ where}$$

$$\begin{aligned} F(t,A) &:= \left(\overline{Kl}'(t)\right)^2 - \left(\overline{X}'(t)^2 + \overline{Y}'(t)^2\right) \\ &= \frac{-(t-1)A^2\left((A^2-1)(t^2+3) - 4t\right)}{4(1+t)^3 t^2\left((1+t)^2 - A^2\right)}. \end{aligned}$$

The last inequality is proven in section A.6 of the appendix. The denominator of $F(t,A)$ is positive since $t \geq \tan\frac{\pi}{4} = 1$ and $A^2 < 2$ holds; see (3.5). Therefore it suffices to show that

$$(A^2-1)(t^2+3) - 4t \leq 0\,. \tag{3.16}$$

We minimize our chances to satisfy (3.16) if A achieves a maximal value greater than 1. Substituting A^2 by $1+\sin(\phi)$, the greatest possible value for A^2, and t by $\tan\frac{\phi}{2}$, the inequality (3.16) holds if

$$2\cos\phi\tan\frac{\phi}{2} \leq 0\,.$$

The details are given in section A.7 of the appendix. The last inequality holds for $\frac{\pi}{2} \leq \phi < \pi$. This proves (3.15) and therefore (3.1) for the curves of section 3.3.

3.5. Opening angles below 90°. So far we have seen that there is a strategy that is competitive with factor $\sqrt{2}$ for opening angles greater than or equal to $\frac{\pi}{2}$. There are already methods to accomplish the task for funnels with opening angles running from an initial angle $\phi_0 < \frac{\pi}{2}$ to an opening angle of $\frac{\pi}{2}$. As was already shown by Semrau [42] and also in López-Ortiz [28], any strategy which achieves a factor $\geq\sqrt{2}$ for all funnels with $\phi_0 \geq \frac{\pi}{2}$ can be adapted to the general case without changing its factor. They suggest a walk along the fixed angular bisector of the current pair v_l and v_r until an opening angle of $\frac{\pi}{2}$ is reached. If the opening angle of $\frac{\pi}{2}$ is reached, one can proceed, for example, with the strategy given in section 3.3. So we are done here.

In the following we show that our idea is universal, and for completeness we consider the case $0 < \phi < \frac{\pi}{2}$ analogously. Looking backwards as in section 3.2 we can assume that there is a strategy which is competitive with factor $\sqrt{2}$ starting at point p_2 with a opening angle $\frac{\pi}{2} \geq \phi_2 > 0$. Again we want to extend the strategy to initial opening angles ϕ_1 at starting points p_1 with $\frac{\pi}{2} \geq \phi_2 > \phi_1 > 0$; see again Figure 3.2. The only difference to the former consideration is that the factor need not vary any longer with respect to the opening angle. The worst-case factor of $\sqrt{2}$ is already in use, and we want to achieve this factor when starting at p_1.

Thus, with the same arguments and notation as in section 3.2, it suffices to show that there is a strategy so that

$$w \le \min(\sqrt{2}\,l_1 - \sqrt{2}\,l_2, \sqrt{2}\,r_1 - \sqrt{2}\,r_2) \tag{3.17}$$

holds between the changes of v_l and v_r as long as the opening angle is smaller than $\frac{\pi}{2}$.

Again, similar to section 3.3, we want to satisfy (3.17) and therefore require that

$$\sqrt{2}\,(l_1 - l_2) = \sqrt{2}\,(r_1 - r_2) \text{ or, equivalently, } (l_2 - r_2) = (l_1 - r_1)\,. \tag{3.18}$$

We consider two cases: For $l_0 = r_0$ we follow the fixed angular bisector in the triangle defined by l_0, r_0, and ϕ_0. In this case, as already stated in the beginning of section 3.4 with $l_0 = r_0$, the equality

$$w = K_{\phi_1} l_1 - K_{\phi_2} l_2 = K_{\phi_1} r_1 - K_{\phi_2} r_2$$

holds. Then (3.17) follows from $K_{\phi_1} \le K_{\phi_2} \le \sqrt{2}$ for $\frac{\pi}{2} \ge \phi_2 > \phi_1 > 0$.

If we start with an initial difference $1 > D := (l_0 - r_0) > 0$ at point s, (3.18) means that we follow the current angular bisector (CAB) of v_l and v_r, and the resulting curve is a hyperbola through s with fix-points v_l and v_r; see Icking, Klein, and Langetepe [20]. For the street problem the strategy CAB was already successfully analyzed for small angles in López-Ortiz and Schuierer [31]. We show that our approach works as well. The transformations of sections 3.3 and 3.4 become much easier since the constant does not depend on ϕ, and all transformations are presented in detail in section A.8 of the appendix. We proceed as before and obtain the coordinates

$$X(\phi) = \frac{D}{2} \cot\frac{\phi}{2} \sqrt{\left(1 + \tan^2\frac{\phi}{2}\right) - D^2},$$
$$Y(\phi) = \frac{1}{2} \cot\frac{\phi}{2} (D^2 - 1).$$

Now $K_\phi l_\phi$ simplifies to

$$\sqrt{2}\,l_\phi = \frac{\sqrt{2}}{D} X(\phi) + \frac{D}{\sqrt{2}}\,.$$

In this case there is no need to simplify the terms by a substitution. In analogy to the previous section it suffices to prove that in (3.10) one integrand dominates the other one; that is,

$$\sqrt{X'(\phi)^2 + Y'(\phi)^2} \le -(K_\phi l_\phi)' \tag{3.19}$$

for all $\frac{\pi}{2} \ge \phi > 0$ and $0 < D < 1$. Altogether it suffices to show

$$F(\phi, D) \ge 0 \quad \text{for all} \quad 0 < \phi \le \frac{\pi}{2} \quad \text{and} \quad 0 < D < 1, \quad \text{where}$$
$$F(\phi, D) := X'(\phi)^2 \left(\frac{2}{D^2} - 1\right) - Y'(\phi)^2$$
$$= \frac{(D^2 - 1)^2 \left(1 - \tan^2\frac{\phi}{2}\right)}{4(\cos\phi - 1)^2 \left(\left(1 + \tan^2\frac{\phi}{2}\right) - D^2\right)}.$$

It remains to be shown that

$$1 - \tan^2 \frac{\phi}{2} \geq 0 \quad \text{and} \quad 1 + \tan^2 \frac{\phi}{2} - D^2 \geq 0$$

holds, which follows from $D^2 < 1$ and $\tan \frac{\phi}{2} \leq 1$ for $\frac{\pi}{2} \geq \phi > 0$. See section A.8 of the appendix for all details.

3.6. The main result. To summarize, our strategy for searching a goal in an unknown street works as follows.

Strategy WCA (worst-case aware). If the initial opening angle is less than $\frac{\pi}{2}$, walk along the current angular bisector of v_l and v_r until a right opening angle is reached.

Depending on the actual parameters ϕ_0, l_0, and r_0, walk along the corresponding curve from section 3.3 until one of v_l and v_r changes. Switch over to the curve corresponding to the new parameters ϕ_1, l_1, and r_1. Continue until the line $t_l\, t_r$ is reached.

THEOREM 3.2. *By using strategy* WCA *we can search a goal in an unknown street with a competitive factor of* $\sqrt{2}$ *at the most. This is optimal.*

The proof can be found in sections 3.1 through 3.5. In Figure 3.7 a complete path of *WCA* inside a street is shown.

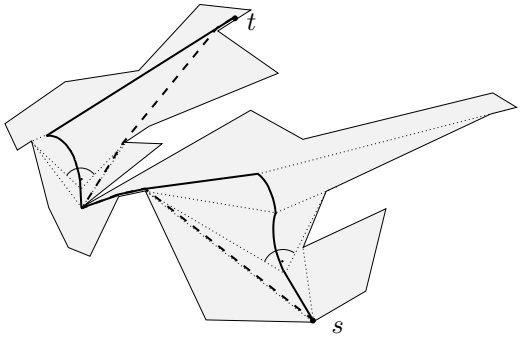


FIG. 3.7. *A street and the path generated by WCA.*

4. Conclusions. We have developed a competitive strategy for walking in streets which guarantees an optimal factor of $\sqrt{2}$ at the most in the worst case, thereby settling an old open problem. Furthermore, the strategy is even better for an initial opening angle $\phi_0 > \frac{\pi}{2}$, in which case an optimal factor $K_{\phi_0} = \sqrt{1 + \sin \phi_0}$ between 1 and $\sqrt{2}$ is achieved.

It would be interesting to see if there are substantially different but also optimal strategies.

Appendix. Formal calculations. This appendix contains the formal calculations needed in the main text. They are presented in a resolution that makes it possible to follow step by step. Nevertheless, the reader might prefer to enter start and target formulae into some math-tool, for example, Maple or Mathematica, and have correctness checked automatically.

A.1. Definition of the circle and the hyperbola. We choose a coordinate system with horizontal axis $v_l\, v_r$, the midpoint being the origin. We scale the coordinate system so that the distance from v_l to v_r equals 1. Let p be the point at a fixed opening angle ϕ on the curve we want to construct. Then two constraints must be met. First, the difference $l(p)-r(p)$ of the length from p to v_l and v_r, correspondingly, must equal $\frac{A}{K_\phi}$. The locus of all such points p is a hyperbola. Second, p sees v_l and v_r at the angle ϕ. The locus of all these points p is a circle; see Figure A.1.

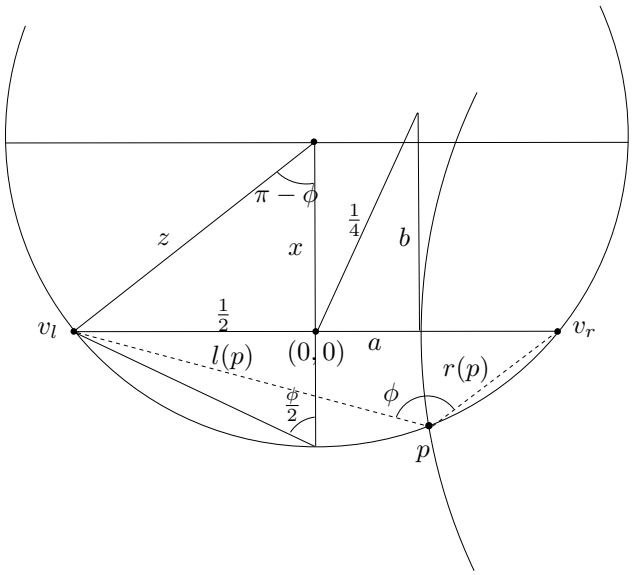


FIG. A.1. *The right arc of the hyperbola defined by v_l, v_r, and $(l(p) - r(p)) = \frac{A}{K_\phi}$ and the circle through v_l and v_r defined by angle ϕ.*

The hyperbola reads

$$\frac{X^2}{a^2} - \frac{Y^2}{b^2} = 1\,,$$

where $2a = (l(p) - r(p)) = \frac{A}{K_\phi}$ and $b^2 + a^2 = e^2 = \frac{1}{4}$ hold. So we have $a^2 = (\frac{A}{2K_\phi})^2$ and $b^2 = \frac{1}{4} - (\frac{A}{2K_\phi})^2$. The circle is defined by

$$X^2 + (Y - x)^2 = z^2\,. \tag{A.1}$$

It remains to compute the parameters of the circle, x and z. From the law of sine we get

$$\frac{z}{\sin\frac{\pi}{2}} = \frac{1}{2\sin(\pi-\phi)} = \frac{1}{2\sin\phi},$$
$$\frac{z-x}{\sin\left(\pi - \frac{\pi}{2} - \frac{\phi}{2}\right)} = \frac{z-x}{\cos\frac{\phi}{2}} = \frac{1}{2\sin\frac{\phi}{2}},$$

and therefore $z = \frac{1}{2\sin\phi}$ and

$$x = z - \frac{1}{2}\cot\frac{\phi}{2} = \frac{1}{2\sin\phi} - \frac{1}{2}\cot\frac{\phi}{2} = \frac{1-2\cos^2\frac{\phi}{2}}{4\sin\frac{\phi}{2}\cos\frac{\phi}{2}} = -\frac{\cot\phi}{2}\,.$$

A.2. Intersection of the circle and the hyperbola. In order to verify the expressions

$$X(\phi) = \frac{A}{2} \cdot \frac{\cot\frac{\phi}{2}}{1+\sin\phi}\sqrt{\left(1+\tan\frac{\phi}{2}\right)^2 - A^2}, \tag{A.2}$$

$$Y(\phi) = \frac{1}{2}\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi} - 1\right), \tag{A.3}$$

we insert them into the equations

$$\frac{X^2}{\left(\frac{A}{2K_\phi}\right)^2} - \frac{Y^2}{\left(\frac{1}{2}\right)^2 - \left(\frac{A}{2K_\phi}\right)^2} = 1, \tag{A.4}$$

$$X^2 + \left(Y + \frac{\cot\phi}{2}\right)^2 = \frac{1}{4\sin^2\phi}. \tag{A.5}$$

For (A.4) we have

$$\begin{aligned}
&\frac{\left(\frac{A}{2}\cdot\frac{\cot\frac{\phi}{2}}{1+\sin\phi}\sqrt{\left(1+\tan\frac{\phi}{2}\right)^2 - A^2}\right)^2}{\left(\frac{A}{2K_\phi}\right)^2} - \frac{\left(\frac{1}{2}\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi}-1\right)\right)^2}{\left(\frac{1}{2}\right)^2 - \left(\frac{A}{2K_\phi}\right)^2} \\
&= \left(\frac{\cot\frac{\phi}{2}}{K_\phi}\right)^2\left(\left(1+\tan\frac{\phi}{2}\right)^2 - A^2\right) - \frac{\cot^2\frac{\phi}{2}\left(\left(\frac{A}{K_\phi}\right)^2 - 1\right)^2}{1-\left(\frac{A}{K_\phi}\right)^2} \\
&= \left(\frac{\cot\frac{\phi}{2}}{K_\phi}\right)^2\left(\left(1+\tan\frac{\phi}{2}\right)^2 - A^2\right) + \cot^2\frac{\phi}{2}\left(\left(\frac{A}{K_\phi}\right)^2 - 1\right) \\
&= \cot^2\frac{\phi}{2}\left(\frac{\left(1+\tan\frac{\phi}{2}\right)^2}{1+\sin\phi} - 1\right) = 1\,.
\end{aligned}$$

The conclusion is true since the identity

$$1+\sin\phi = 1 + \frac{2\tan\frac{\phi}{2}}{1+\tan^2\frac{\phi}{2}} = \frac{\left(1+\tan\frac{\phi}{2}\right)^2}{1+\tan^2\frac{\phi}{2}} \tag{A.6}$$

holds.

For proving (A.5) we argue as follows:

$$\begin{aligned}
&\left(\frac{A}{2}\cdot\frac{\cot\frac{\phi}{2}}{1+\sin\phi}\sqrt{\left(1+\tan\frac{\phi}{2}\right)^2 - A^2}\right)^2 + \left(\frac{1}{2}\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi}-1\right) + \frac{\cot\phi}{2}\right)^2 \\
&= \left(\frac{A}{2}\cdot\frac{\cot\frac{\phi}{2}}{1+\sin\phi}\right)^2\left(\left(1+\tan\frac{\phi}{2}\right)^2 - A^2\right)
\end{aligned}$$

$$
\begin{aligned}
&+\left(\frac{1}{2}\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi}-1\right)\right)^2+\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi}-1\right)\frac{\cot\phi}{2}+\left(\frac{\cot\phi}{2}\right)^2\\
&=\left(\frac{A}{2}\cdot\frac{\cot\frac{\phi}{2}}{1+\sin\phi}\right)^2\left(1+\tan\frac{\phi}{2}\right)^2\\
&\quad+\left(\frac{1}{2}\cot\frac{\phi}{2}\right)^2\left(-2\frac{A^2}{1+\sin\phi}+1\right)+\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi}-1\right)\frac{\cot\phi}{2}+\left(\frac{\cot\phi}{2}\right)^2\\
&=\left(\frac{\cot\frac{\phi}{2}}{2}-\frac{\cot\phi}{2}\right)^2+\frac{A^2\cot^2\frac{\phi}{2}}{4(1+\sin\phi)}\left(\frac{\left(1+\tan\frac{\phi}{2}\right)^2}{1+\sin\phi}-2+2\frac{\cot\phi}{\cot\frac{\phi}{2}}\right)\\
&=\frac{1}{4\sin^2\phi}+\frac{A^2\cot^2\frac{\phi}{2}}{4(1+\sin\phi)}\left(\tan^2\frac{\phi}{2}+1-2+\frac{1-\tan^2\frac{\phi}{2}}{\tan\frac{\phi}{2}}\tan\frac{\phi}{2}\right)\\
&=\frac{1}{4\sin^2\phi}+\frac{A^2\cot^2\frac{\phi}{2}}{4(1+\sin\phi)}\cdot 0=\frac{1}{4\sin^2\phi}\,.
\end{aligned}
$$

Here we made use of (A.6) and the identities

$$
\left(\frac{\cot\frac{\phi}{2}}{2}-\frac{\cot\phi}{2}\right)^2=\frac{1}{4}\left(\frac{\sin\phi}{1-\cos\phi}-\frac{\cos\phi}{\sin\phi}\right)^2=\frac{1}{4}\frac{1}{\sin^2\phi}
$$

and

$$
\cot\phi=\frac{1-\tan^2\frac{\phi}{2}}{2\tan\frac{\phi}{2}}\,.
$$

A.3. Representation of l_ϕ. Considering the hyperbola (see Figure 3.5), we have

$$
\begin{aligned}
l_\phi &= \sqrt{\left(X(\phi)+\frac{1}{2}\right)^2+Y^2(\phi)}\\
&\overset{(3.6)}{=} \sqrt{\left(X(\phi)+\frac{1}{2}\right)^2-X^2(\phi)-\left(\frac{1}{2}\right)^2+\left(\frac{A}{2K_\phi}\right)^2+\frac{\left(\frac{1}{2}\right)^2}{\left(\frac{A}{2K_\phi}\right)^2}X^2(\phi)}\\
&= \sqrt{\left(\frac{K_\phi}{A}\right)^2X^2(\phi)+X(\phi)+\left(\frac{A}{2K_\phi}\right)^2}\\
&= \sqrt{\left(\frac{K_\phi}{A}X(\phi)+\frac{A}{2K_\phi}\right)^2}=\frac{K_\phi}{A}X(\phi)+\frac{A}{2K_\phi}\,.
\end{aligned}
$$

A.4. Applying the substitution $t=\tan\frac{\phi}{2}$. Applying (A.6) and $t=\tan\frac{\phi}{2}$ we have

$$
K_\phi=\sqrt{1+\sin\phi}=\sqrt{\frac{(1+t)^2}{1+t^2}}=:\overline{K}(t)\,. \tag{A.7}
$$

A straightforward application of (A.7) and $t = \tan\frac{\phi}{2}$ to (A.2) and (A.3) leads to

$$X(\phi) = \frac{A}{2t\overline{K}^2(t)}\sqrt{(1+t)^2 - A^2} = \frac{A(1+t^2)\sqrt{(1+t)^2-A^2}}{2t(1+t)^2} =: \overline{X}(t)\ ,$$
$$Y(\phi) = \left(\frac{A^2}{\overline{K}^2(t)} - 1\right)\frac{1}{2t} \qquad = \quad \frac{A^2(1+t^2) - (1+t)^2}{2t(1+t)^2} \quad =: \overline{Y}(t)\ .$$

Using the representations of $\overline{X}(t)$ and $\overline{K}(t)$ we conclude that

$$K_\phi l_\phi = \frac{K_\phi}{A}X(\phi) + \frac{A}{2K_\phi} = \frac{\overline{K}^2(t)}{A}\overline{X}(t) + \frac{A}{2} = \frac{\sqrt{(1+t)^2-A^2}}{2t} + \frac{A}{2} =: \overline{Kl}(t)$$

holds.

A.5. Computing the derivatives in t. We apply simple derivation rules:

$$\begin{aligned}
-\overline{Kl}'(t) &= -\left(\frac{(1+t)}{2t\sqrt{(1+t)^2-A^2}} - \frac{\sqrt{(1+t)^2-A^2}}{2t^2}\right)\\
&= -\left(\frac{t(1+t) - (1+t)^2 + A^2}{2t^2\sqrt{(1+t)^2-A^2}}\right)\\
&= \frac{(1+t) - A^2}{2t^2\sqrt{(1+t)^2-A^2}},\\
\overline{Y}'(t) &= \left(\frac{A^2(1+t^2)}{2t(1+t)^2} - \frac{1}{2t}\right)'\\
&= \frac{2tA^2}{2t(1+t)^2} + A^2(1+t^2)\left(-\frac{(1+t)^2 + 2(1+t)t}{2(1+t)^4t^2}\right) + \frac{1}{2t^2}\\
&= \frac{(1+t)^3 - A^2\left(-2t^2(1+t) + (1+t^2)(1+3t)\right)}{2(1+t)^3t^2}\\
&= \frac{(1+t)^3 - A^2\left((1+t)^3 - 4t^2\right)}{2(1+t)^3t^2},\\
\overline{X}'(t) &= \left(\frac{A(1+t^2)}{2t(1+t)^2}\sqrt{(1+t)^2-A^2}\right)'\\
&= \left(\frac{2tA}{2t(1+t)^2} + A(1+t^2)\left(-\frac{(1+t)^2+2(1+t)t}{2(1+t)^4t^2}\right)\right)\sqrt{(1+t)^2-A^2}\\
&\quad + \frac{1+t}{\sqrt{(1+t)^2-A^2}}\frac{A(1+t^2)}{2t(1+t)^2}\\
&= \frac{-A((1+t)^3-4t^2)}{2(1+t)^3t^2\sqrt{(1+t)^2-A^2}}((1+t)^2-A^2) + \frac{t(1+t)^2A(1+t^2)}{2(1+t)^3t^2\sqrt{(1+t)^2-A^2}}
\end{aligned}$$

$$= \frac{-A((1+t)^3-4t^2)((1+t)^2-A^2)+t(1+t)^2A(1+t^2)}{2(1+t)^3t^2\sqrt{(1+t)^2-A^2}}$$

$$= \frac{A(A^2\left((1+t)^3-4t^2\right))-A\left(((1+t)^3-4t^2)(1+t)^2-t(1+t)^2(1+t^2)\right)}{2(1+t)^3t^2\sqrt{(1+t)^2-A^2}}$$

$$= \frac{A(A^2\left((1+t)^3-4t^2\right))-A((1+t)^5-(1+t)^2(4t^2+t+t^3))}{2(1+t)^3t^2\sqrt{(1+t)^2-A^2}}$$

$$= \frac{A(A^2\left((1+t)^3-4t^2\right))-A((1+t)^5-(1+2t+t^2)(4t^2+t+t^3))}{2(1+t)^3t^2\sqrt{(1+t)^2-A^2}}$$

$$= \frac{A(A^2\left((1+t)^3-4t^2\right))-A((1+t)^5-t-6t^2-10t^3-6t^4-t^5)}{2(1+t)^3t^2\sqrt{(1+t)^2-A^2}}$$

$$= \frac{A\left(A^2\left((1+t)^3-4t^2\right)-1-4t-4t^2+t^4\right)}{2(1+t)^3t^2\sqrt{(1+t)^2-A^2}}.$$

A.6. How to get $F(t, A)$. We will make use of the following identities:

$$2(t^2-2t-1)-(4t^2-(1+t)^3)+2(1+t) = (1+t)(t^2+1), \tag{A.8}$$

$$(t^2-2t-1)(1+t)^2 = (t^4-1-4t-4t^2), \tag{A.9}$$

$$\begin{aligned}(t-1)(t+3)t^2 &= -3t^2+2t^3+t^4\\ &= -2(1+t)^3-(t^2-2t-1)^2\\ &\quad +(1+t)^2-8t^2(1+t)+2(1+t)^4,\end{aligned} \tag{A.10}$$

$$(1+t)^3+(4t^2-(1+t)^3)(t^2+1) = t^2(1-t)(t^2+3)\ . \tag{A.11}$$

$$\begin{aligned}F(t,A) &:= \overline{Kl}'(t)^2-\overline{X}'(t)^2-\overline{Y}'(t)^2\\ &= \left(\frac{(1+t)-A^2}{2t^2\sqrt{(1+t)^2-A^2}}\right)^2-\left(\frac{A\left(A^2\left((1+t)^3-4t^2\right)-1-4t-4t^2+t^4\right)}{2(1+t)^3t^2\sqrt{(1+t)^2-A^2}}\right)^2\\ &\quad -\left(\frac{(1+t)^3-A^2\left((1+t)^3-4t^2\right)}{2(1+t)^3t^2}\right)^2\\ &= \frac{\left(((1+t)-A^2)(1+t)^3\right)^2}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)}-\frac{\left(A\left(A^2\left((1+t)^3-4t^2\right)-1-4t-4t^2+t^4\right)\right)^2}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)}\\ &\quad -\frac{\left((1+t)^3-A^2\left((1+t)^3-4t^2\right)\right)^2\left((1+t)^2-A^2\right)}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)}\\ &= \frac{(1+t)^6A^4-2(1+t)^7A^2+(1+t)^8}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)}\end{aligned}$$

$$
\begin{aligned}
&- \frac{((1+t)^3-4t^2)^2A^6 + 2(t^4-1-4t-4t^2)((1+t)^3-4t^2)A^4}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)} \\
&\qquad - \frac{(t^4-1-4t-4t^2)^2A^2}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)} \\
&- \frac{-((1+t)^3-4t^2)^2A^6 + (-2(1+t)^3(4t^2-(1+t)^3) + (4t^2-(1+t)^3)^2(1+t)^2)A^4}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)} \\
&\qquad - \frac{(-(1+t)^6 + 2(1+t)^5(4t^2-(1+t)^3))A^2 + (1+t)^8}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)}
\end{aligned}
$$

$$
\begin{aligned}
\overset{(A.9)}{=}\ & A^4 \frac{(1+t)^6 + 2(t^2-2t-1)(1+t)^2(4t^2-(1+t)^3)}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)} \\
&+ A^4 \frac{-(-2(1+t)^3(4t^2-(1+t)^3) + (4t^2-(1+t)^3)^2(1+t)^2)}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)} \\
&+ A^2 \frac{-2(1+t)^7 - (t^2-2t-1)^2(1+t)^4 - (-(1+t)^6 + 2(1+t)^5(4t^2-(1+t)^3))}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)} \\
=\ & A^4 \frac{(1+t)^6 + (1+t)^2(4t^2-(1+t)^3)(2(t^2-2t-1) - (4t^2-(1+t)^3) + 2(1+t))}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)} \\
&+ A^2 \frac{-2(1+t)^7 - (1+t)^4(t^2-2t-1)^2 + (1+t)^6 - 2(1+t)^5(4t^2-(1+t)^3))}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)} \\
\overset{(A.8)}{=}\ & A^4 \frac{(1+t)^6 + (1+t)^2(4t^2-(1+t)^3)(1+t)(t^2+1)}{\left(2(1+t)^3t^2\right)^2\left((1+t)^2-A^2\right)} \\
&+ A^2 \frac{(1+t)\left(-2(1+t)^3 - (t^2-2t-1)^2 + (1+t)^2 - 8t^2(1+t) + 2(1+t)^4\right)}{(1+t)^3(2t^2)^2\left((1+t)^2-A^2\right)} \\
\overset{(A.10)}{=}\ & A^4 \frac{(1+t)^3 + (4t^2-(1+t)^3)(t^2+1)}{(1+t)^3(2t^2)^2\left((1+t)^2-A^2\right)} + A^2 \frac{(1+t)(t-1)(t+3)t^2}{(1+t)^3(2t^2)^2\left((1+t)^2-A^2\right)} \\
\overset{(A.11)}{=}\ & A^4 \frac{-t^2(t-1)(t^2+3)}{(1+t)^3(2t^2)^2\left((1+t)^2-A^2\right)} + A^2 \frac{(1+t)(t-1)(t+3)}{4t^2(1+t)^3\left((1+t)^2-A^2\right)} \\
=\ & \frac{-(t-1)A^2\left((A^2-1)(t^2+3) - 4t\right)}{4(1+t)^3t^2\left((1+t)^2-A^2\right)}.
\end{aligned}
$$

A.7. Transformation of the nominator of $F(t, A)$.

$$
\sin\phi\left(\tan^2\frac{\phi}{2}+3\right) - 4\tan\frac{\phi}{2} = \sin\phi\left(\frac{2\tan\frac{\phi}{2}}{\sin\phi}+2\right) - 4\tan\frac{\phi}{2}
$$

$$
\begin{aligned}
&= 2\sin\phi - 2\tan\frac{\phi}{2}\\
&= 2\sin\frac{\phi}{2}\left(2\cos\frac{\phi}{2} - \frac{1}{\cos\frac{\phi}{2}}\right)\\
&= 2\sin\frac{\phi}{2}\left(\frac{2\cos^2\frac{\phi}{2}-1}{\cos\frac{\phi}{2}}\right)\\
&= 2\sin\frac{\phi}{2}\left(\frac{\cos\phi}{\cos\frac{\phi}{2}}\right) = 2\cos\phi\tan\frac{\phi}{2}.
\end{aligned}
$$

A.8. The simple case $\phi < \frac{\pi}{2}$. In order to obtain

$$
\begin{aligned}
X(\phi) &= \frac{D}{2}\cot\frac{\phi}{2}\sqrt{\left(1+\tan^2\frac{\phi}{2}\right) - D^2},\\
Y(\phi) &= \frac{1}{2}\cot\frac{\phi}{2}(D^2-1),
\end{aligned}
$$

and

$$
K_\phi l_\phi := \sqrt{2}\, l_\phi = \frac{\sqrt{2}}{D}X(\phi) + \frac{D}{\sqrt{2}},
$$

we simply replace $\frac{A}{K_\phi}$ by D in (3.8), (3.9), and (3.11). For (3.8), which corresponds to $X(\phi)$, we have to make use of identity (A.6).

Since we did not make use of a substitution here, the derivatives in ϕ are given as follows:

$$
\begin{aligned}
-(K_\phi l_\phi)' &= -\frac{\sqrt{2}}{D}X'(\phi),\\
Y'(\phi) &= -\frac{1}{4\sin^2\frac{\phi}{2}}(D^2-1)\\
&= \frac{(D^2-1)}{2(\cos\phi-1)},\\
X'(\phi) &= \left(\frac{D}{2}\cot\frac{\phi}{2}\sqrt{\left(1+\tan^2\frac{\phi}{2}\right)-D^2}\right)'\\
&= \frac{D}{2(\cos\phi-1)}\sqrt{\left(1+\tan^2\frac{\phi}{2}\right)-D^2} + \frac{D}{2}\cot\frac{\phi}{2}\left(\frac{-\tan\frac{\phi}{2}\frac{1}{\cos^2\frac{\phi}{2}}}{2\sqrt{\left(1+\tan^2\frac{\phi}{2}\right)-D^2}}\right)\\
&= \frac{\frac{D}{(\cos\phi-1)}\left(\left(1+\tan^2\frac{\phi}{2}\right)-D^2\right) - \frac{D}{2}\frac{1}{\cos^2\frac{\phi}{2}}}{2\sqrt{\left(1+\tan^2\frac{\phi}{2}\right)-D^2}}\\
&= \frac{\frac{D(1-D^2)}{(\cos\phi-1)} + \frac{D\tan^2\frac{\phi}{2}}{(\cos\phi-1)} - \frac{D}{2}\frac{1}{\cos^2\frac{\phi}{2}}}{2\sqrt{\left(1+\tan^2\frac{\phi}{2}\right)-D^2}}
\end{aligned}
$$

$$= \frac{D(1-D^2)}{2(\cos\phi - 1)\sqrt{\left(1+\tan^2\frac{\phi}{2}\right) - D^2}}.$$

It remains to compute $F(\phi, D)$.

$$\begin{aligned}
F(\phi, D) &:= (K_\phi l_\phi)^2 - (X'(\phi))^2 - (Y'(\phi))^2 \\
&= X'(\phi)^2 \left(\frac{2}{D^2} - 1\right) - Y'(\phi)^2 \\
&= \frac{D^2(1-D^2)^2\left(\frac{2}{D^2}-1\right)}{4(\cos\phi-1)^2\left(\left(1+\tan^2\frac{\phi}{2}\right)-D^2\right)} - \frac{(D^2-1)^2}{4(\cos\phi-1)^2} \\
&= \frac{(D^2-1)^2\left((2-D^2) - \left(\left(1+\tan^2\frac{\phi}{2}\right)-D^2\right)\right)}{4(\cos\phi-1)^2\left(\left(1+\tan^2\frac{\phi}{2}\right)-D^2\right)} \\
&= \frac{(D^2-1)^2\left(1-\tan^2\frac{\phi}{2}\right)}{4(\cos\phi-1)^2\left(\left(1+\tan^2\frac{\phi}{2}\right)-D^2\right)}.
\end{aligned}$$

REFERENCES

[1] S. Alpern and S. Gal, *The Theory of Search Games and Rendezvous*, Kluwer Academic, Dordrecht, The Netherlands, 2002.

[2] R. Baeza-Yates, J. Culberson, and G. Rawlins, *Searching in the plane*, Inform. and Comput., 106 (1993), pp. 234–252.

[3] R. Bellman, *Problem* 63-9*, SIAM Rev., 5 (1963), p. 274.

[4] P. Berman, *On-line searching and navigation*, in Online Algorithms: The State of the Art, A. Fiat and G. Woeginger, eds., Springer-Verlag, Berlin, 1998, pp. 232–241.

[5] M. Blom, S. O. Krumke, W. de Paepe, and L. Stougie, *The online-TSP against fair adversaries*, INFORMS J. Comput., 13 (2001), pp. 138–148.

[6] A. Blum, P. Raghavan, and B. Schieber, *Navigating in unfamiliar geometric terrain*, SIAM J. Comput., 26 (1997), pp. 110–137.

[7] C. Bröcker and A. López-Ortiz, *Position-independent street searching*, in Algorithms and Data Structures (Vancouver, BC, 1999), Lecture Notes in Comput. Sci. 1663, Springer-Verlag, Berlin, 1999, pp. 241–252.

[8] C. Bröcker and S. Schuierer, *Searching rectilinear streets completely*, in Algorithms and Data Structures (Vancouver, BC, 1999), Lecture Notes in Comput. Sci. 1663, Springer-Verlag, Berlin, 1999, pp. 98–109.

[9] S. Carlsson and B. J. Nilsson, *Computing vision points in polygons*, Algorithmica, 24 (1999), pp. 50–75.

[10] G. Das, P. Heffernan, and G. Narasimhan, *LR-visibility in polygons*, Comput. Geom. Theory Appl., 7 (1997), pp. 37–57.

[11] P. Dasgupta, P. P. Chakrabarti, and S. C. DeSarkar, *A new competitive algorithm for agent searching in unknown streets*, in Foundations of Software Technology and Theoretical Computer Science (Hyderabad, 1996), Lecture Notes in Comput. Sci. 1180, Springer-Verlag, Berlin, 1996, pp. 147–155.

[12] A. Datta, C. A. Hipke, and S. Schuierer, *Competitive searching in polygons: Beyond generalised streets*, in Algorithms and Computations (Cairns, 1995), Lecture Notes in Comput. Sci. 1004, Springer-Verlag, Berlin, 1995, pp. 32–41.

[13] A. Datta and C. Icking, *Competitive searching in a generalized street*, in Proceedings of the 10th Annual ACM Symposium on Computational Geometry, ACM Press, New York, 1994, pp. 175–182.

[14] A. Datta and C. Icking, *Competitive searching in a generalized street*, Comput. Geom. Theory Appl., 13 (1999), pp. 109–120.

[15] A. Fiat and G. Woeginger, eds., *Online Algorithms: The State of the Art*, Lecture Notes in Comput. Sci. 1442, Springer-Verlag, Berlin, 1998.

[16] S. Gal, *Search Games*, Math. Sci. Engrg. 149, Academic Press, New York, 1980.
[17] S. K. Ghosh and S. Saluja, *Optimal on-line algorithms for walking with minimum number of turns in unknown streets*, Comput. Geom. Theory Appl., 8 (1997), pp. 241–266.
[18] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel, *The polygon exploration problem*, SIAM J. Comput., 31 (2001), pp. 577–600.
[19] C. Icking, *Motion and Visibility in Simple Polygons*, Ph.D. thesis, Department of Computer Science, FernUniversität Hagen, Germany, 1994.
[20] C. Icking, R. Klein, and E. Langetepe, *Searching for the Kernel of a Polygon: A Competitive Strategy Using Self-approaching Curves*, Tech. Rep. 211, Department of Computer Science, FernUniversität Hagen, Germany, 1997.
[21] C. Icking, R. Klein, and E. Langetepe, *An optimal competitive strategy for walking in streets*, in Proceedings of the 16th Symposium on Theoretical Aspects in Computer Science, Lecture Notes in Comput. Sci. 1563, Springer-Verlag, Berlin, 1999, pp. 110–120.
[22] C. Icking, R. Klein, and L. Ma, *How to look around a corner*, in Proceedings of the 5th Canadian Information Processing Society Congress, University of Waterloo, Waterloo, Canada, 1993, pp. 443–448.
[23] C. Icking, A. López-Ortiz, S. Schuierer, and I. Semrau, *Going Home through an Unknown Street*, Tech. Rep. 228, Department of Computer Science, FernUniversität Hagen, Germany, 1998.
[24] R. Klein, *Walking an unknown street with bounded detour*, in Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 304–313.
[25] R. Klein, *Walking an unknown street with bounded detour*, Comput. Geom. Theory Appl., 1 (1992), pp. 325–351.
[26] J. M. Kleinberg, *On-line search in a simple polygon*, in Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (Arlington, VA, 1994), ACM Press, New York, 1994, pp. 8–15.
[27] E. Kranakis and A. Spatharis, *Almost optimal on-line search in unknown streets*, in Proceedings of the 9th Canadian Conference on Computational Geometry, Kingston, ON, Canada, 1997, pp. 93–99.
[28] A. López-Ortiz, *On-line Target Searching in Bounded and Unbounded Domains*, Ph.D. thesis, University of Waterloo, Waterloo, Canada, 1996.
[29] A. López-Ortiz and S. Schuierer, *Going home through an unknown street*, in Algorithms and Data Structures (Kingston, ON, 1995), Lecture Notes in Comput. Sci. 955, Springer-Verlag, Berlin, 1995, pp. 135–146.
[30] A. López-Ortiz and S. Schuierer, *Generalized streets revisited*, in Proceedings of the 4th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 1136, Springer-Verlag, Berlin, 1996, pp. 546–558.
[31] A. López-Ortiz and S. Schuierer, *Walking streets faster*, in Proceedings of the 5th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 1097, Springer-Verlag, Berlin, 1996, pp. 345–356.
[32] A. López-Ortiz and S. Schuierer, *The Exact Cost of Exploring Streets with CAB*, tech. rep., Institut für Informatik, Universität Freiburg, Germany, 1998.
[33] A. López-Ortiz and S. Schuierer, *Lower Bounds for Searching on Generalized Streets*, tech. rep., University of New Brunswick, Canada, 1998.
[34] A. López-Ortiz and S. Schuierer, *Online parallel heuristics and robot searching under the competitive framework*, in Proceedings of the 8th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 2368, Springer-Verlag, Berlin, 2002, pp. 260–269.
[35] J. S. B. Mitchell, *Geometric shortest paths and network optimization*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North-Holland, Amsterdam, 2000, pp. 633–701.
[36] T. Ottmann, S. Schuierer, and C. A. Hipke, *Kompetitive Analyse für Online-Algorithmen: Eine kommentierte Bibliographie*, Tech. Rep. 61, Institut für Informatik, Universität Freiburg, Germany, 1994.
[37] N. S. V. Rao, S. Kareti, W. Shi, and S. S. Iyengar, *Robot Navigation in Unknown Terrains: Introductory Survey of Non-heuristic Algorithms*, Tech. Rep. ORNL/TM-12410, Oak Ridge National Laboratory, Oak Ridge, TN, 1993.
[38] J.-R. Sack and J. Urrutia, eds., *Handbook of Computational Geometry*, North-Holland, Amsterdam, 2000.
[39] S. Schuierer, *Lower bounds in on-line geometric searching*, in Proceedings of the 11th International Symposium on Fundamentals of Computation Theory, Lecture Notes in Comput. Sci. 1279, Springer-Verlag, Berlin, 1997, pp. 429–440.

[40] S. Schuierer and I. Semrau, *An optimal strategy for searching in unknown streets*, in Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1563, Springer-Verlag, Berlin, 1999, pp. 121–131.
[41] R. Seidel, *personal communications via Rolf Klein*, 1999.
[42] I. Semrau, *Analyse und experimentelle Untersuchung von Strategien zum Finden eines Ziels in Straßenpolygonen*, diploma thesis, FernUniversität Hagen, Germany, 1996.
[43] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.
[44] L. H. Tseng, P. Heffernan, and D. T. Lee, *Two-guard walkability of simple polygons*, Internat. J. Comput. Geom. Appl., 8 (1998), pp. 85–116.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 487–501

LEARNING A HIDDEN MATCHING*

NOGA ALON†, RICHARD BEIGEL‡, SIMON KASIF§,
STEVEN RUDICH¶, AND BENNY SUDAKOV‖

Abstract. We consider the problem of learning a matching (i.e., a graph in which all vertices have degree 0 or 1) in a model where the only allowed operation is to query whether a set of vertices induces an edge. This is motivated by a problem that arises in molecular biology. In the deterministic nonadaptive setting, we prove a $(\frac{1}{2}+o(1))\binom{n}{2}$ upper bound and a nearly matching $0.32\binom{n}{2}$ lower bound for the minimum possible number of queries. In contrast, if we allow randomness, then we obtain (by a randomized, nonadaptive algorithm) a much lower $O(n \log n)$ upper bound, which is best possible (even for randomized fully adaptive algorithms).

Key words. matchings in graphs, combinatorial search problems, genome sequencing, finite projective planes

AMS subject classifications. 05C85, 05C35, 68W20, 68Q17

DOI. 10.1137/S0097539702420139

1. Introduction.

This paper is motivated by an important and timely problem in computational biology that arises in whole-genome shotgun sequencing. Shotgun sequencing is a high throughput technique that has resulted in the sequencing of a large number of bacterial genomes (over 70 at the time of this writing), as well as *Drosophila* (fruit fly) and mouse genomes and the celebrated human genome (at Celera). In all such projects, we are left with a collection of contigs (long DNA sequences) that for various biological or computational reasons cannot be assembled with even the best sequence assembly algorithms. The contigs must be ordered and oriented and the *gaps* between them must be sequenced using slower, more tedious methods. When the number of gaps is small (e.g., less than ten) biologists often use polymerase chain reaction (PCR). This technique initiates a set of "bidirectional molecular walks" along the gaps in the sequence; these walks are facilitated by PCR.

In order to initiate the molecular walks biologists use primers.[1] Primers are

*Received by the editors December 20, 2002; accepted for publication (in revised form) November 24, 2003; published electronically March 5, 2004. A preliminary version of this paper, *Combinatorial Identification of Hidden Matchings with Applications to Whole Genome Sequencing*, appeared in the Proceedings of the 2002 IEEE FOCS.

http://www.siam.org/journals/sicomp/33-2/42013.html

†Institute for Advanced Study, Princeton, NJ 08540, and Department of Mathematics, Tel Aviv University, Tel Aviv 69978, Israel (nogaa@post.tau.ac.il). This author's research was supported in part by a State of New Jersey grant, by a USA Israeli BSF grant, by a grant from the Israel Science Foundation, and by the Hermann Minkowski Minerva Center for Geometry at Tel Aviv University.

‡Institute for Advanced Study, Princeton, NJ 08540, and Department of Computer and Information Sciences, Temple University, 1805 N. Broad St., Philadelphia, PA 19122 (beigel@cis.temple.edu, http://www.cis.temple.edu/∼beigel). This author's research was supported in part by the NSF under grants CCR-0049019 and CCR-9877150.

§Department of Biomedical Engineering, Boston University, 44 Cummington St., Boston, MA 02215 (kasif@bu.edu).

¶Department of Computer Science, Carnegie Mellon University, Wean Hall 4111, Pittsburgh, PA 15213 (rudich@cs.cmu.edu).

‖Department of Mathematics, Princeton University, Princeton, NJ 08540, and Institute for Advanced Study, Princeton, NJ 08540 (bsudakov@math.princeton.edu). This author's research was supported in part by NSF grants DMS-0106589 and CCR-9987845 and by the State of New Jersey.

[1]Primers are short preconstructed single-stranded polynucleotide chains to which new deoxyribonucleotides can be "appended" by DNA polymerase.

designed so that they bind to unique (with respect to the entire DNA sequence) templates occurring at the end of each contig. A primer (at the right temperature and concentration) anneals to the designated unique DNA substring and promotes copying of the template starting from the primer binding site, initiating a one-directional walk along the gap in the DNA sequence. A PCR reaction occurs, and can be observed as a DNA ladder, when two primers that bind to positions on two ends of the same gap are placed in the same test tube.

If we are left with N contigs, the combinatorial (exhaustive) PCR technique tests all possible pairs (quadratically many) of $2N$ primers by placing two primers per tube with the original uncut DNA strand. PCR products can be detected using gels, or they can be read using sequencing technology or DNA mass-spectometry. When the number of gaps is large, the quadratic number of PCR experiments is prohibitive, so primers are pooled using $K > 2$ primers per tube; this technique is called multiplex PCR.[2] Our paper provides optimal strategies for pooling the primers to minimize the number of biological experiments needed in the gap-closing process.

Our gap-closing problem can be stated more generally as follows. We are given a set of chemicals, a guarantee that each chemical reacts with at most one of the others (because only primers on opposite sides of the same gap create a reaction), and an experimental mechanism to determine whether a reaction occurs when several chemicals are combined in a test tube. We wish to determine which pairs of chemicals react with each other with a minimum number of experiments.

Our problem can be modeled as the problem of identifying or learning a hidden matching given a vertex set and an allowed query operation ([9, 2]; see [5, 6] for an alternative formulation). A vertex will represent a chemical, an edge of the matching will represent a reaction, and a query will represent checking for a reaction when a set of chemicals are combined in a test tube. Let $V = \{1, 2, \ldots, n\}$. We wish to identify an unknown (not necessarily perfect) matching M on V by asking a small number of queries of the form

$$Q_F: \quad \text{does } F \text{ contain at least one edge of } M? \tag{1.1}$$

where F is a subset of V. This problem is of interest even in the deterministic, fully nonadaptive case. We say that a family $\mathcal{F}$ of subsets of V *solves the matching problem* on V if for any two distinct matchings M_1 and M_2 on V there is at least one $F \in \mathcal{F}$ that contains an edge of one of the matchings and does not contain any edge of the other. Obviously, any such family enables us to learn an unknown matching deterministically and nonadaptively by asking the questions Q_F defined in (1.1) for each $F \in \mathcal{F}$.

Our objective is to estimate the minimum possible cardinality of a family that solves the matching problem on a set of n vertices. Toward this end, we will generalize the matching problem to finding matchings contained in a graph H (not necessarily complete) and produce an H for which we can solve the matching problem with a family of size roughly half the number of edges of H. By applying a partitioning theorem of Wilson, we can then solve the matching problem on n vertices with a family of size $(\frac{1}{2} + o(1))\binom{n}{2}$.

[2]The earliest reference to multiplex PCR is [3]. Since then hundreds of papers report using the multiplex PCR technique to answer a diverse set of questions in molecular biology. Multiplex PCR using a simple, nonoptimal pooling strategy has recently been applied successfully at The Institute for Genomic Research (TIGR) to close gaps in a number of genomes, including *Streptococcus* [9].

We show that our construction is tight up to a constant factor, as stated in the following theorem.

THEOREM 1.1. *For every $n > 2$, every family $\mathcal{F}$ that solves the matching problem on n vertices satisfies*

$$|\mathcal{F}| \geq \frac{49}{153}\binom{n}{2}.$$

The proof of the lower bound is presented in sections 3 and 4.

Next we consider randomized nonadaptive algorithms. In contrast to the 1-round deterministic case we produce, somewhat surprisingly, an $O(n \log n)$ solution in this model. This solution is asymptotically optimal up to a constant factor, because of the information-theoretic $\Omega(n \log n)$ lower bound, even if we do not restrict the number of rounds. We believe that the sharp difference between deterministic and randomized nonadaptive algorithms here is remarkable; while one can hardly beat the trivial $\binom{n}{2}$ bound in the deterministic case, the randomized fully nonadaptive algorithm is already as efficient (up to a constant factor) as the best possible fully adaptive algorithm for the problem. Moreover, the same technique shows that a hidden copy of any sparse graph, that is, a graph with a linear number of edges in which all degrees are $O(\sqrt{n})$, can be found, with high probability, in a 1-round randomized algorithm making only $O(n \log n)$ queries.

Finally we present a deterministic k-round algorithm for learning a hidden matching which makes $O(kn^{1+1/(2(k-1))}\text{polylog}n)$ queries. Our deterministic 2-round algorithm asks $\frac{5}{4}n^{3/2}(1 + o(1))$ queries of size at most $n^{1/4}$ each. This is optimal up to a factor of $5/4$ among *all* algorithms that make queries of size at most $n^{1/4}$, which may be useful in view of practical limitations on multiplexing. For $k \geq 3$ our algorithms are based on a coloring lemma for projective planes that may be interesting in its own right.

Our techniques combine combinatorial and probabilistic tools with results about graph decomposition and about the existence of certain designs. Throughout the paper, we omit all floor and ceiling signs, whenever these are not crucial. All logarithms are in base 2, unless otherwise specified.

2. Related work. In an earlier paper with Fortnow and Apaydin [2] we obtained a randomized, adaptive algorithm that solves the matching problem in 8 rounds with an expected number of approximately $0.72n \log_2 n$ queries. Our results here improve the number of rounds to 1 in the randomized case. (This comes at the cost of doubling the number of queries. If we allow 2 rounds, we can, in fact, keep the total number of queries to be roughly $0.72n \log_2 n$.) We further show here that in the 1-round, deterministic case, far more queries are needed, though some saving over the trivial algorithm is possible.

Grebinski and Kucherov [5, 6] consider the problem of finding a Hamilton cycle. They obtain an $O(n \log n)$ *adaptive* algorithm. (They also have an $O(n)$ purely nonadaptive solution using more powerful queries, i.e., queries that report the *number* of edges induced by a set of vertices. See [4, 7] as well.) Using our methods here we can show that $\Omega(n^2)$ queries are needed for finding a Hamilton cycle in the deterministic nonadaptive case in our model. A similar $\Omega(n^2)$ lower bound can be proved for the problem of determining the *number* of edges of a hidden matching, as well as for the problem of finding a hidden copy of any given bounded degree graph with $\Omega(n)$ edges.

3. Sparse families. A family of sets $\mathcal{A} = \{A_1, \ldots, A_k\}$ is *sparse* if there is a collection of pairwise disjoint pairs of members of $V = \bigcup_{i=1}^{k} A_i$ such that each A_i

contains at least one of the pairs. Therefore, $\mathcal{A}$ is sparse iff there is a matching on V such that the answer to each question Q_A for $A \in \mathcal{A}$ is "yes." It is easy to see that any set of more than $(p^2+p+1)/2$ lines in a projective plane of order p (in which each line is of size $p+1$) is not sparse, and our results here will imply that every family consisting of at most $0.32\binom{m+2}{2}$ sets, each of size at least m, is sparse.

For a family $\mathcal{F}$ of subsets define the *t-weight* of the family, denoted $w_t(\mathcal{F})$, as follows:

$$w_t(\mathcal{F}) = \sum_{F \in \mathcal{F}} \frac{1}{\binom{|F|+t}{2}}.$$

The 2-weight is simply called the *weight* and is denoted, for short, by $w(\mathcal{F})$. The main lemma of this section is the following.

LEMMA 3.1. *Every family $\mathcal{F}$ of sets whose weight is at most 49/153 is sparse.*

Proof. If $\mathcal{F}$ contains a set of size 1, then $w(\mathcal{F}) \geq 1/3 > 49/153$. Thus we may and will assume that all sets in $\mathcal{F}$ are of size at least 2. Let $M \in \mathcal{F}$ be a set of minimum cardinality, $|M| = m$.

For a pair of distinct elements p, q, define

$$\mathcal{F}(p,q) = \{F - \{p,q\} : F \in \mathcal{F}, \{p,q\} \not\subseteq F\}.$$

Note that if we pick the pair $\{p,q\}$ as a member of the matching we are trying to construct to show that $\mathcal{F}$ is sparse, then the members of $\mathcal{F}(p,q)$ are precisely those that will have to be handled by the rest of the matching. This suggests proving the following claim.

Claim. There exists a pair of distinct elements p, q of M such that $w(\mathcal{F}(p,q)) \leq w(\mathcal{F})$.

To prove the claim we choose p, q randomly and uniformly among all pairs of distinct members of M and show that the expected value $\mathbb{E}(w(\mathcal{F}(p,q)))$ is at most $w(\mathcal{F})$.

Henceforth F will denote an element of $\mathcal{F} - \{M\}$. Let $\kappa(F)$ denote $|F \cap M|$. We have that $\mathbb{E}(w(\mathcal{F}(p,q)))$ equals

$$= w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \sum_{F \neq M} \left(\frac{\kappa(F)(m-\kappa(F))}{\binom{m}{2}} \left(\frac{1}{\binom{|F|+1}{2}} - \frac{1}{\binom{|F|+2}{2}} \right) - \frac{\binom{\kappa(F)}{2}}{\binom{m}{2}} \frac{1}{\binom{|F|+2}{2}} \right)$$

$$= w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \sum_{k<m} \sum_{\kappa(F)=k} \left(\frac{k(m-k)}{\binom{m}{2}} \left(\frac{1}{\binom{|F|+1}{2}} - \frac{1}{\binom{|F|+2}{2}} \right) - \frac{\binom{k}{2}}{\binom{m}{2}} \frac{1}{\binom{|F|+2}{2}} \right)$$

$$= w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m}{2}} \sum_{k<m} \sum_{\kappa(F)=k} \left(k(m-k) \left(\frac{\binom{|F|+2}{2}}{\binom{|F|+1}{2}} - 1 \right) - \binom{k}{2} \right) \frac{1}{\binom{|F|+2}{2}}$$

$$= w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m}{2}} \sum_{k<m} \sum_{\kappa(F)=k} \left(\frac{2k(m-k)}{|F|} - \binom{k}{2} \right) \frac{1}{\binom{|F|+2}{2}}$$

$$\leq w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m}{2}} \sum_{k<m} \sum_{\kappa(F)=k} \left(\frac{2k(m-k)}{m} - \binom{k}{2} \right) \frac{1}{\binom{|F|+2}{2}}$$

$$= w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m}{2}} \sum_{k<m} \left(\frac{2k(m-k)}{m} - \binom{k}{2} \right) \sum_{\kappa(F)=k} \frac{1}{\binom{|F|+2}{2}}$$

$$= w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m}{2}} \sum_{k<m} \mu(m,k) \sum_{\kappa(F)=k} \frac{1}{\binom{|F|+2}{2}},$$

where we define $\mu(m,k) = \frac{2k(m-k)}{m} - \binom{k}{2}$. For all m, we have $\mu(m,0) = 0$, $\mu(m,1) = 2 - 2/m$, $\mu(m,2) = 3 - 8/m$, and $k \geq 2 \Rightarrow \mu(m,k) \leq \mu(m,2)$. Thus $\mu(m,k)$ is maximized at $k = 1$ or $k = 2$. Define $\mu(m) = \max_{k<m} \mu(m,k)$. Now we have

$$\begin{aligned}
\mathbb{E}(w(\mathcal{F}(p,q))) &\leq w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m}{2}} \sum_{k<m} \mu(m,k) \sum_{\kappa(F)=k} \frac{1}{\binom{|F|+2}{2}} \\
&\leq w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m}{2}} \sum_{k<m} \mu(m) \sum_{\kappa(F)=k} \frac{1}{\binom{|F|+2}{2}} \\
&= w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m}{2}} \mu(m) \sum_{k<m} \sum_{\kappa(F)=k} \frac{1}{\binom{|F|+2}{2}} \\
&= w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m}{2}} \mu(m) \sum_{F \neq M} \frac{1}{\binom{|F|+2}{2}} \\
&= w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m}{2}} \mu(m) \left(w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} \right) \\
&= w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} + \frac{1}{\binom{m+2}{2}} \frac{(m+2)(m+1)}{m(m-1)} \mu(m) \left(w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} \right).
\end{aligned}$$

Thus it suffices to prove that

$$\frac{(m+2)(m+1)}{m(m-1)} \mu(m) \left(w(\mathcal{F}) - \frac{1}{\binom{m+2}{2}} \right) \leq 1$$

or, equivalently, that

$$w(\mathcal{F}) \leq \frac{m(m-1)}{(m+2)(m+1)\mu(m)} + \frac{1}{\binom{m+2}{2}}.$$

As noted above, $\mu(m)$ is either $\mu(m,1)$ or $\mu(m,2)$. In the first case, we have

$$\frac{m(m-1)}{(m+2)(m+1)\mu(m)} + \frac{1}{\binom{m+2}{2}} = \frac{m^2+4}{2(m+2)(m+1)} \geq \frac{13}{40}$$

for $m > 1$ (with equality at $m = 3$). In the second case, we have

$$\frac{m(m-1)}{(m+2)(m+1)\mu(m)} + \frac{1}{\binom{m+2}{2}} = \frac{1}{3} \frac{m^2(m-1) + 6m - 16}{(m+2)(m+1)(m-8/3)} \geq \frac{49}{153}$$

for $m > 2$ (with equality at $m = 16$). By assumption, $w(\mathcal{F}) \leq 49/153 < 13/40$, completing the proof of the claim.

By repeatedly applying the claim we get smaller and smaller families of sets whose weights remain bounded by 49/153. This process must terminate with a matching that captures all members of $\mathcal{F}$, showing that $\mathcal{F}$ is sparse and completing the proof of the lemma. □

4. The proof of the main result for the fully nonadaptive case. In this short section we present the proof of Theorem 1.1. We need the following simple fact.

LEMMA 4.1. *Let $\mathcal{F}$ be a family of subsets of $V = \{1, 2, \ldots, n\}$ that solves the matching problem on V. Then, for every two distinct $a, b \in V$, the family $\mathcal{F}_{a,b} = \{F - \{a,b\} : F \in \mathcal{F} \text{ and } \{a,b\} \subseteq F\}$ is not sparse.*

Proof. Assume that this is false, and that $\mathcal{F}_{a,b}$ is sparse for some $a, b \in V$. Then there is a matching M in $V - \{a, b\}$ so that each member of $\mathcal{F}_{a,b}$ contains an edge of M. But then the answers to each question Q_F with $F \in \mathcal{F}$ for the two matchings M and $M \cup \{a, b\}$ are identical, contradicting the fact that $\mathcal{F}$ solves the matching problem. □

Proof of Theorem 1.1. Let $\mathcal{F}$ be a family of subsets of $V = \{1, 2, \ldots, n\}$ that solves the matching problem on V. Let a, b be any pair of distinct vertices in V. We know by Lemma 4.1 that the family $\mathcal{F}_{a,b} = \{F - \{a, b\} : F \in \mathcal{F} \text{ and } \{a, b\} \subseteq F\}$ is not sparse. Therefore, by Lemma 3.1,

$$\sum_{F \in \mathcal{F}, \{a,b\} \subseteq F} \frac{1}{\binom{|F|}{2}} = \sum_{F' \in \mathcal{F}_{a,b}} \frac{1}{\binom{|F'|+2}{2}} > \frac{49}{153}.$$

We can now assign, for each $F \in \mathcal{F}$ a weight of $1/\binom{|F|}{2}$ to each pair of distinct elements $a, b \in F$. The total weight distributed in this way is precisely $|\mathcal{F}|$, as the total contribution of each $F \in \mathcal{F}$ is 1. On the other hand, the total weight assigned to each pair $a, b \in V$ is at least $\frac{49}{153}$, implying that $|\mathcal{F}| \geq \frac{49}{153}\binom{n}{2}$, as needed. □

Note that the same proof supplies a $\frac{49}{153}\binom{n}{2}$ lower bound for the number of queries in any 1 round deterministic algorithm that determines the *number* of edges of a hidden matching on n vertices.

5. Other hidden graphs. In this section we show how to extend the methods described in the previous two sections and obtain a lower bound for the number of queries needed to find a hidden copy of a member of a given family of graphs with certain properties. Throughout the section, we make no attempt to optimize the absolute constants in our various estimates.

Let $\mathcal{H}$ be a family of labeled graphs on the set $V = \{1, 2, \ldots, n\}$, and suppose $\mathcal{H}$ is closed under isomorphism. Thus, for example, $\mathcal{H}$ may be the set of all Hamilton cycles on V, or all matchings on V, or all perfect matchings on V. Our objective is to learn a hidden copy of some member of $\mathcal{H}$ by asking a small number of queries Q_F as given in (1.1). We say that a family $\mathcal{F}$ solves the $\mathcal{H}$-problem if for any two distinct members H_1 and H_2 in $\mathcal{H}$ there is at least one $F \in \mathcal{F}$ that contains an edge of one of the graphs H_i and does not contain any edge of the other. Obviously, any such family enables us to learn an unknown member of $\mathcal{H}$ deterministically and nonadaptively by asking the questions Q_F defined in (1.1) for each $F \in \mathcal{F}$.

THEOREM 5.1. *There exists an absolute constant $c > 0$ such that the following holds. Let $\mathcal{H}$ be a family of graphs on V, closed under isomorphism, and suppose that there are two distinct graphs $H_1, H_2 \in \mathcal{H}$ and a set of vertices $D \subset V$, $|D| = d$ satisfying the following:*

(i) *The graphs obtained from H_1 and from H_2 by omitting all edges connecting two vertices of D are identical; and*

(ii) *there is a matching of at least pn edges in H_1 which contains no vertices of D (clearly this matching is also a matching in H_2).*

Then, if $1/p > d$, every family $\mathcal{F}$ that solves the $\mathcal{H}$-problem satisfies $|\mathcal{F}| \geq c\frac{p^2}{d^2}\binom{n}{2}$.

Note that this result provides an $\Omega(n^2)$ lower bound for the problem of learning a *perfect* matching or a Hamilton cycle, and, more generally, the problem of learning a hidden copy of any fixed, bounded-degree graph with $\Omega(n)$ edges. It also provides an $\Omega(n^2)$ lower bound for the problem of finding a hidden copy of a vertex disjoint union of a clique of size $n - 3$ and a single edge, but *not* for the problem of finding a hidden copy of a vertex disjoint union of a clique of size $n - 2$ and a single edge (and

indeed it is easy to see that $O(n)$ queries suffice for the latter problem).

A family $\mathcal{F}$ of subsets of V is *p-sparse* if there is a collection of at most pn pairwise disjoint pairs of members of V such that each $F \in \mathcal{F}$ contains at least one of the pairs. Therefore, $\mathcal{F}$ is p-sparse iff there is a matching on V consisting of at most pn edges such that the answer to each query Q_F for $F \in \mathcal{F}$ is "yes."

LEMMA 5.2. *There is an absolute constant $c_1 > 0$ such that every family $\mathcal{F}$ of subsets of V of weight at most $c_1 p^2$ is p-sparse.*

Proof. Let V_1 be a random subset of V obtained by picking each $v \in V$, randomly and independently, to lie in V_1 with probability p. As the expected size of V_1 is pn, it follows that with probability at least one-half, its size is at most $2pn$. For each set $F \in \mathcal{F}$, the expected size of $F \cap V_1$ is clearly $p|F|$, and as this size is a binomial random variable, it follows, by the standard estimates for binomial distributions (see, e.g., [1, Appendix A, Theorem A.1.13], that the probability that it is less than $p|F|/2$ does not exceed $e^{-p|F|/8}$. Since the weight of $\mathcal{F}$ is at most $c_1 p^2$ for some (small) c_1, we conclude that each set in $\mathcal{F}$ is of size at least, say, $100/p$. As $e^{-px/8} < 1/p^2 \binom{x+2}{2}$ for all $x > 100/p$, it follows that the probability that there is some set $F \in \mathcal{F}$ such that $|F \cap V_1| < p|F|/2$ is smaller than $\frac{w(\mathcal{F})}{p^2} \leq c_1 < 1/2$. Therefore, there exists a set $V_1 \subset V$ of cardinality at most $2pn$ so that $|F \cap V_1| \geq p|F|/2$ for all $F \in \mathcal{F}$. As $|F| > 100/p$ for each $F \in \mathcal{F}$, this implies that the weight of the family $\mathcal{F}_1 = \{F \cap V_1 : F \in \mathcal{F}\}$ is at most, say, $\frac{8}{p^2} w(\mathcal{F}) \leq 8c_1 < 49/153$ (assuming c_1 is sufficiently small). By Lemma 3.1, $\mathcal{F}_1$ is sparse, and as $|V_1| \leq 2pn$, it follows that $\mathcal{F}$ is p-sparse. □

LEMMA 5.3. *Let $\mathcal{H}$, V, n, d, and p be as in Theorem 5.1. Let $\mathcal{F}$ be a family of subsets of $V = \{1, 2, \ldots, n\}$ that solves the $\mathcal{H}$-problem. Then, for every subset $D \subset V$, $|D| = d$, the family $\mathcal{F}_D = \{F - D : F \in \mathcal{F},\ |F \cap D| \geq 2\}$ is not p-sparse.*

Proof. Assume this is false, and suppose $\mathcal{F}_D$ is p-sparse for some $D' \subset V$, $|D'| = d$. Then there is a matching M of size at most pn in $V - D'$ so that each member of $\mathcal{F}_{D'}$ contains an edge of M. The matching M can be completed to a graph with no edges in D' which is isomorphic to the graph obtained from H_1 (or H_2) by omitting the edges inside D, where the isomorphism maps D onto D'. It is now possible to extend this graph to a copy of H_1, or to a copy of H_2, only by adding edges inside D'. But then the answers to each question Q_F with $F \in \mathcal{F}$ for these two distinct members of $\mathcal{H}$ are identical, contradicting the fact that $\mathcal{F}$ solves the $\mathcal{H}$-problem. □

Proof of Theorem 5.1. Let $\mathcal{F}$ be a family of subsets of $V = \{1, 2, \ldots, n\}$ that solves the $\mathcal{H}$-problem. By Lemmas 5.2 and 5.3, for every set D consisting of d vertices of V, the weight of the family $\mathcal{F}_D = \{F - D : F \in \mathcal{F}, |F \cap D| \geq 2\}$ is at least $c_1 p^2$. We claim that

$$\sum_{F \in \mathcal{F}, |F \cap D| \geq 2} \frac{1}{\binom{|F|}{2}} \geq c_2 p^2$$

for every D as above (and for an appropriately chosen $c_2 > 0$). Indeed, if there is a set $F \in \mathcal{F}$ of size at most, say, $10/p$, that intersects D in at least 2 elements, this follows immediately. Otherwise, by the assumption that $1/p > d$,

$$\frac{1}{\binom{|F|-d+2}{2}} \leq 2 \frac{1}{\binom{|F|}{2}},$$

and the claim follows from the fact that the weight of $\mathcal{F}_D$ is at least $c_1 p^2$.

If D is a random subset of d vertices of V, then for every $F \in \mathcal{F}$, the probability that $|D \cap F| \geq 2$ is at most $\binom{d}{2}\binom{|F|}{2}/\binom{n}{2}$. It follows that the expected value of the

random variable

$$\sum_{F\in\mathcal{F},|F\cap D|\geq 2}\frac{1}{\binom{|F|}{2}}$$

is at most $\binom{d}{2}|\mathcal{F}|/\binom{n}{2}$, and as this random variable is always at least c_2p^2, the desired result follows. □

6. An upper bound for the fully nonadaptive case. In this section we show how to design families of size $(\frac{1}{2}+o(1))\binom{n}{2}$ to solve the matching problem on n vertices. It will be helpful if we first generalize the matching problem. We say that a family $\mathcal{F}$ of cliques contained in G *solves the matching problem* on G if for any two distinct matchings contained in G there is at least one clique in $\mathcal{F}$ that contains an edge of one of the matchings and does not contain any edge of the other. (Note, for example, that if G is triangle free, then $\mathcal{F}$ must be a set of edges.) The matching problem on n vertices is thus the same as the matching problem on K_n. Let $f(G)$ denote the size of the smallest family that solves the matching problem on G.

Throughout this section, let $E(H)$ denote the edge set of a graph H, and let $\gcd(H)$ denote the greatest common divisor of the degrees of all vertices in H.

THEOREM 6.1 (Wilson [10]). *For every graph H there exists a constant N such that for all $n\geq N$, K_n is the union of $\binom{n}{2}/|E(H)|$ pairwise edge-disjoint graphs isomorphic to H iff $\binom{n}{2}$ is divisible by $|E(H)|$ and $n-1$ is divisible by $\gcd(H)$.*

COROLLARY 6.2. *For every fixed graph H,*

$$f(K_n)\leq\frac{f(H)}{|E(H)|}\binom{n}{2}+O(n).$$

Furthermore, for fixed H, the solution to the matching problem for K_n is constructive.

Proof. By Wilson's theorem, K_n is the union of $\binom{n}{2}/|E(H)|$ graphs isomorphic to H. Solve the matching problem in each of those graphs with a family of size $f(H)$. □

We say that a family $\mathcal{F}$ determines the status of an edge e if for every pair of matchings M_1, M_2 such that $e\in M_1$ and $e\notin M_2$ there is at least one clique $\mathcal{F}$ that contains an edge of one of the matchings and does not contain any edge of the other.

The reader may easily verify the following lemma.

LEMMA 6.3 (two-thirds). *Let a,b,c,x be four distinct vertices. Assume that $\mathcal{F}$ determines the status of $\{a,b\}$ and $\{b,c\}$. If $\mathcal{F}$ contains the two triangles $\{a,b,x\}$ and $\{b,c,x\}$, then $\mathcal{F}$ also determines the status of $\{a,x\}$, $\{b,x\}$, and $\{c,x\}$.*

We note in passing that one may easily apply the two-thirds lemma to obtain a solution of size $\frac{2}{3}\binom{n}{2}+O(n)$.

DEFINITION 6.4 (HEX_s^+). *Let $s\geq 1$. Tile a hexagon having side length s with unit equilateral triangles. Add one more vertex Z and edges from Z to every vertex in the tiling. Call the resulting graph HEX_s^+.*

The tiling above, depicted in Figure 6.1, contains $v=3s^2+3s+1$ vertices, $e=9s^2+3s$ edges, and $f=6s^2$ triangles. Therefore the graph HEX_s^+ contains $v+e=12s^2+6s+1$ edges. We solve the matching problem on HEX_s^+ with the following tests:

Tetrahedra $T\cup\{Z\}$ for every triangle T in the tiling.

Boundary every boundary edge of the tiling, and every edge from Z to a point on the boundary.

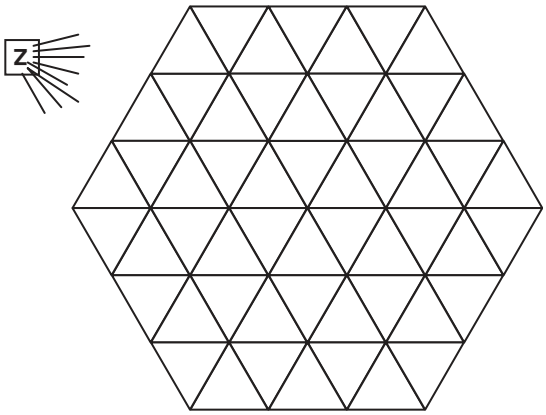


FIG. 6.1. *The graph* HEX_s^+.

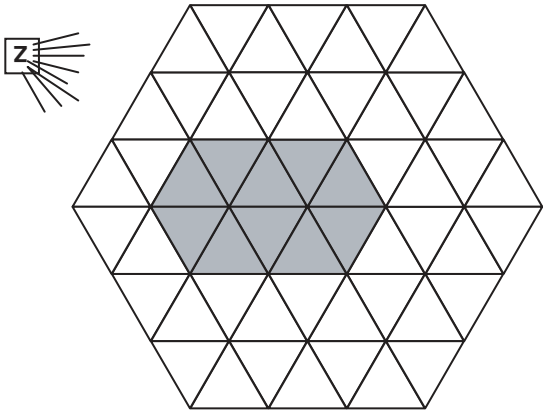


FIG. 6.2. *Locate* Z*'s mate (if it exists).*

As a warm-up, let us see why these tests suffice, assuming that Z is unmatched. In this case, the tetrahedra queries are equivalent to triangles, and we just apply the two-thirds lemma repeatedly, starting at the boundary of the tiling.

Now let us see why these tests suffice in general. Try the following cases in order.

Case 1. For some Y on the boundary of the tiling, $ZY \in M$. This edge is tested, so we know $ZY \in M$. Finish as in the warm-up.

Case 2. For some Y in the interior of the tiling, $ZY \in M$. In this case all six tests containing Y say yes (see Figure 6.2). Call that the 6-triangle property for Y. If only one point has the 6-triangle property, then we know that that point is matched to Z. If three distinct points have the 6-triangle property, it is easy to check that they must be adjacent in a straight line, and the middle one must be matched to Z. Consequently, no more than three points can have the 6-triangle property. In either of those subcases we finish as in the warm-up (see Figures 6.3 and 6.4).

If exactly two points have the 6-triangle property, then they must be adjacent; we proceed as in the warm-up until we reach the 10 triangles containing those two points. With the aid of the tests already performed, a simple case analysis tells us

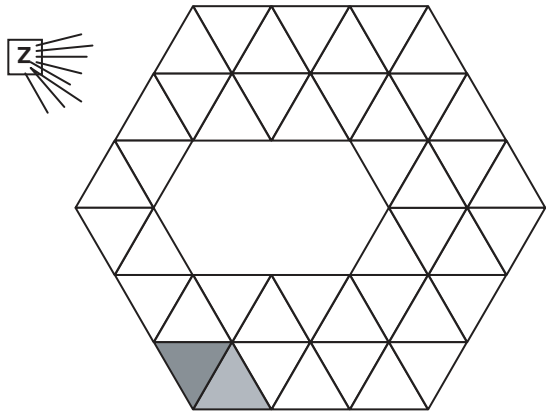


FIG. 6.3. *Test corners using Lemma* 6.3.

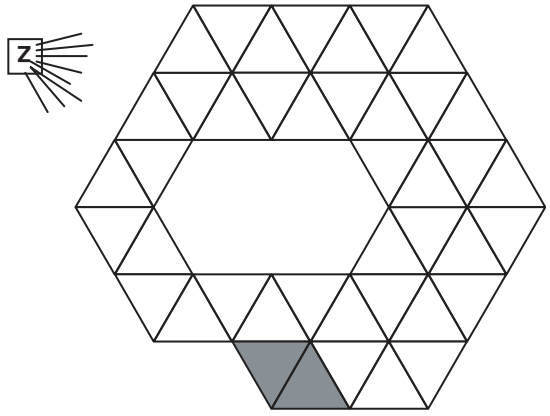


FIG. 6.4. *Continue, using Lemma* 6.3.

which point is matched to Z.

Case 3. For all Y in the tiling, $ZY \notin M$. Then no point has the 6-triangle property, so we know that Z is unmatched. Proceed as in the warm-up.

The total number of tests is $f + 12s = 6s^2 + 12s$. The ratio of tests to edges is

$$(6s^2 + 12s)/(12s^2 + 6s + 1) = \frac{1}{2} + \frac{18s - 1}{24s^2 + 12s + 2}.$$

Combining this with the corollary of Wilson's theorem, we see that $f(K_n) = (1/2 + O(1/s))\binom{n}{2}$. Letting s be a slowly growing function of n, we obtain the following.

COROLLARY 6.5.

$$f(K_n) \leq \left(\frac{1}{2} + o(1)\right)\binom{n}{2}.$$

7. Probabilistic nonadaptive algorithms. In this section we present a very efficient randomized algorithm for the matching problem. The simplest version of the

algorithm queries $bn \log n$ random subsets of size $c\sqrt{n}$ each. The analysis below shows that for an appropriate choice of b and c, the algorithm solves, with high probability, the matching problem in one round. Since we believe that this algorithm and some of its variants may be of practical interest, we do make here some efforts to optimize the absolute constant obtained in the estimate for the total number of queries. It turns out that in order to improve the constant, it is better to ask the queries according to randomly shifted (modified) projective planes. The details follow. For completeness, we include a brief description of the relevant properties of finite projective planes. A projective plane $PG(2,p)$ of order p consists of p^2+p+1 points and p^2+p+1 lines. Each line contains exactly $p+1$ points and each point lies in precisely $p+1$ lines. Every two distinct points are contained in a unique common line, and every two distinct lines intersect in a unique common point. Such a plane exists for every prime power p and can be constructed by the following simple algebraic description. Let B denote the set of all nonzero vectors $\bar{x} = (x_0, x_1, x_2)$ of length 3 over the finite field $GF(p)$, and define an equivalence relation on B by calling two vectors equivalent if one is a multiple of the other by an element of the field. The points of $PG(2,p)$ as well as the hyperplanes can be represented by the equivalence classes of B with respect to this relation, where a point $\bar{x} = (x_0, x_1, x_2)$ lies in the hyperplane $\bar{y} = (y_0, y_1, y_2)$ iff their inner product $\langle x, y\rangle = x_0y_0 + x_1y_1 + x_2y_2$ over $GF(p)$ is zero. More details about projective planes can be found, for example, in [8].

Procedure RPP: Testing according to a random projective plane. We assume now that $n = p^2+p+1$ for some prime p. Testing according to a random projective plane consists of the following: Randomly permute our n vertices and identify them with the points of the projective plane P of order p. Perform one test for each line.

Now consider a pair (x, y). Exactly one line in P contains x and y. The probability that that line contains no (other) edge of M is at least the value of this probability when the matching is perfect and xy is not a matching edge. It is not difficult to see that in this case, the probability is precisely

$$\frac{(n-4)(n-6)\dots(n-2p)}{(n-2)(n-3)\dots(n-p)}.$$

Indeed, the number is the number of ways to choose an ordered set of the other $p-1$ points of the line (besides x and y) without containing any matching edge, as after i points (including x and y) have already been chosen, there are $n-2i$ possibilities for choosing the next point, which has to be different from the chosen points and their mates. The denominator is the total number of possibilities for choosing an ordered set of $p-1$ points. The last expression is at least

$$e^{-(1-o(1))\frac{p^2}{2n}} = e^{-1/2}(1-o(1)).$$

Testing according to $d \log n$ random projective planes. Perform procedure RPP $d \log n$ times independently in parallel, for some real number d. The probability that every line containing x and y contains an edge of M (other than possibly (x, y)) is at most $\pi(d) = ((1-e^{-1/2})^d(1+o(1)))^{\log n}$. If we choose $d = (1+o(1)) \ln 2/\ln(1/(1-e^{-1/2})) \approx 0.74$, then $\pi(d) \le 1/n$.

Consequently, those tests suffice to identify all but $n/2$ nonmatching edges on average. Those remaining nonedges and all matching edges can be identified in a second round with only n tests. Thus we have a 2-round algorithm for the matching problem that makes an expected number of approximately $0.74n \log n$ tests and makes no errors.

If we choose d twice as large, i.e., $d = 2\ln 2/\ln(1/(1-e^{-1/2})) + \epsilon$ for some arbitrarily small ϵ (say $d \approx 1.49$), then $\pi(d) = o(1/n^2)$. Consequently, those tests suffice to identify all nonmatching edges with high probability. Once we have identified all nonedges, the same reasoning shows that all the matching edges are identified with high probability as well. Thus we have a 1-round algorithm for the matching problem that makes an expected number of approximately $1.49n\log n$ tests and makes no errors.

Point doubling. These constants can be improved. When every vertex is in the same number of tests, the ideal test size is approximately $\sqrt{(2\ln 2)n}$, but there are no designs like the projective plane with sets of size greater than $\sqrt{n}$. Fortunately, we do not need every pair of points to belong to exactly one set. It suffices to construct designs in which every pair of points belongs to at least one set, provided that we do not generate too many sets in the process. This can be accomplished rather easily by randomly "doubling" some of the points in the projective plane. We double a point x by adding a new point x' to each line that contains x.

To be precise, we assume now that $n = \lceil(2\ln 2)m\rceil$, where $m = p^2+p+1$ and p is prime. We start with the projective plane of order p and double $\lceil(2\ln 2 - 1)m\rceil$ randomly chosen points. This results in n points. We still have $m \approx n/(2\ln 2)$ lines. By standard estimates, with high probability each of those lines has approximately $2\ln 2\sqrt{m} \approx \sqrt{(2\ln 2)n}$ points.

Now, let us look at a single design and a single pair of points x, y. Consider a "line" containing x and y. (If x and y are not the duplicates of a single point, there is one such line; otherwise, there are $p+1$ such lines, and then it suffices to consider one of them.) Let $t \approx \sqrt{(2\ln 2)n}$ be the number of points on this line. The probability that it contains no (other) edge of M—besides, possibly, xy—is, by the same reasoning described above, at least

$$\frac{(n-4)(n-6)\dots(n-2t+2)}{(n-2)(n-3)\dots(n-t+1)} = e^{-(1+o(1))\frac{t^2}{2n}} = (1/2)(1+o(1)).$$

Now take $d\log n$ random projective planes with doubled points of the type above. The probability that every "line" containing x and y contains an edge of M (other than possibly (x,y)) is at most $\pi'(d) = \big((1/2)^d\big(1+o(1)\big)\big)^{\log n}$. Thus $\pi'(1+o(1)) = 1/n$ and $\pi'(2+o(1)) = 1/n^2$. Since each design contains approximately $n/(2\ln 2)$ "lines," we obtain the following.

THEOREM 7.1. *The matching problem on n vertices can be solved by probabilistic algorithms with the following parameters:*

- 2 *rounds and* $(1/(2\ln 2))n\log n(1+o(1)) \approx 0.72n\log n$ *tests;*
- 1 *round and* $(1/\ln 2)n\log n(1+o(1)) \approx 1.44n\log n$ *tests.*

Note that the algorithms make no errors in the sense that when we get the answers we know which edges are matching edges and which are not. With high probability, we get all the information in the 1-round algorithm; in the rare event we do not, we know it, and we can make an additional set of queries for all the edges whose status has not been determined. In the 2-round algorithm we always get all the information, but with positive probability we will have to ask more than n queries in the second round.

Note also that the algorithms described here can be easily modified to find a hidden copy of any graph with $O(n)$ edges and with maximum degree $O(\sqrt{n})$ in one randomized round, using $O(n\log n)$ queries.

8. Deterministic k-round algorithms. In this section we present *deterministic* k-round algorithms that make at most $O(n^{1+1/(2(k-1))}\text{polylog}n)$ queries per round. In the special case $k = 2$, we do not need the polylog$n$ factor. All of our deterministic algorithms are constructive.

A lemma about the chromatic number of the graph consisting of all edges contained in half the lines of a projective plane will allow us to reduce the general problem to a bipartite matching problem. The lemma, which may be interesting in its own right, is proved by considering the eigenvalues of the plane's incidence matrix.

LEMMA 8.1 (coloring). *Let P be a finite projective plane with n points. Obtain Q by deleting at least $n/2$ of P's lines. Let $G = (V, E)$, where V is P's point set and E consists of all pairs (x, y) such that there is a line in Q containing both x and y. Then G is $\sqrt{n}\ln n(1 + o(1))$ colorable using color classes of size less than $\sqrt{n}$. Furthermore such a coloring can be found in time polynomial in n.*

Proof. The basic idea in the proof of this lemma is as follows. Consider a set B of points. On average, a line will contain about $|B|/\sqrt{n}$ points. We will show that if B is not very small, then most lines contain at least half that number of points. This will allow us to greedily choose our color classes from among the lines that were deleted from P. We need the following lemma, whose proof is essentially identical to that of Lemma 9.2.4 of [1].

LEMMA 8.2. *Let $G = (U, V; E)$ be a d-regular bipartite graph with classes of vertices U and V of size n each. Let $A = (A_{u,v} : u \in U, v \in V)$ be the (bipartite) adjacency matrix of G given by $A_{u,v} = 1$ iff $uv \in E$, and $A_{u,v} = 0$ otherwise. Suppose, further, that every eigenvalue of A^tA except the largest (which is d^2) is at most λ^2. Then, for every $B \subseteq V$,*

$$\sum_{u\in U}\left(|N(u)\cap B| - d\frac{|B|}{n}\right)^2 \le \lambda^2|B|\left(1 - \frac{|B|}{n}\right).$$

Let P be a projective plane of order p. Thus it has $n = p^2 + p + 1$ points. Let G be the incidence graph of P, i.e., the bipartite graph with classes of vertices U and V, with $|U| = |V| = n$, in which V is the set of points and U is the set of lines, where uv is an edge iff the line u contains the point v. If A is the adjacency matrix of G, then A^tA is a matrix in which all diagonal entries are $p+1$ and all other entries are 1. Consequently, the largest eigenvalue of A^tA is $(p+1)^2$ and all its other eigenvalues are equal to p. It follows that for every set of points $B \subset U$, we can bound the number of lines v containing less than $\frac{d|B|}{2n}$ points of B by the above lemma. Namely,

$$\begin{aligned}\left|\left\{v \in V : |N(v)\cap B| < \frac{d}{2}\frac{|B|}{n}\right\}\right| &< \frac{4\lambda^2}{d^2}\frac{n^2}{|B|}\\ &= \frac{4p}{(p+1)^2}\frac{n^2}{|B|}\\ &\le \frac{4n^{3/2}}{|B|}.\end{aligned}$$

Therefore, if $|B| > 10\sqrt{n}$, then every set consisting of $0.4n$ lines contains a line that contains at least $\frac{\sqrt{n}}{2}\frac{|B|}{n} = \frac{|B|}{2\sqrt{n}}$ elements of B. Now we are in a position to prove the following.

COROLLARY 8.3. *Every set S consisting of at least $0.4n$ lines contains a subset consisting of at most $\sqrt{n}\ln n$ lines covering all but at most $10\sqrt{n}$ points.*

Proof. Initially, let $B = V$. As long as $|B| \geq 10\sqrt{n}$, we may choose a line in S that contains at least a $\frac{1}{2\sqrt{n}}$ fraction of the points in B and then remove those points from B. After at most $2\sqrt{n}\ln[n/(10\sqrt{n})] < \sqrt{n}\ln n$ iterations, we will have reduced B to size at most $10\sqrt{n}$. □

To complete the proof of the coloring lemma, we take our set of lines to be the ones deleted from P. Our color classes are the $\sqrt{n}\ln n$ lines promised by the preceding corollary as well as the $10\sqrt{n}$ singletons not covered by those lines. If a point belongs to more than one of those lines, then we can choose its color class arbitrarily from among them. □

LEMMA 8.4 (first bipartite). *Assume that M is a nonempty matching on V. Let V be the disjoint union of L and R, where $|R| \geq 2$, and assume we know that neither L nor R contains an edge of M. We can learn M with a k-round algorithm that makes at most $|M||L|^{1/k}\log|R|$ tests per round.*

Proof. The proof is by induction on k. If $k = 1$, then we can perform a parallel binary search for each element of L to find its match, if any, in R. The number of tests performed is $|L|\log|R|$.

Now let $k \geq 2$ and assume we have a $(k-1)$-round algorithm that makes at most $|M||L|^{1/(k-1)}\log|R|$ tests per round. Let $t = |L|^{1/k}$. Partition L into t pieces $L_1, \ldots, L_t$ of size $|L|/t$. In round 1, test $L_i \cup R$ for each i. At most $|M|$ of those sets can contain an edge, say $L_{i_1} \cup R, \ldots, L_{i_m} \cup R$, where $m \leq |M|$. Apply the inductive hypothesis to the matching problems on those m sets. Let e_j denote the number of edges in L_{i_j}. The number of tests per round is at most

$$\sum_j e_j(|L|/t)^{1/(k-1)}\log|R| = |M||L|^{1/k}\log|R| \qquad \square$$

LEMMA 8.5 (second bipartite). *Assume that M is a nonempty matching on V. Let V be the disjoint union of L and R, and assume we know that neither L nor R contains an edge of M. Let c be a real number such that $0 < c < 1$, $c|L|^{1/k} \geq 1$. Let $k \geq 2$. We can learn M with a k-round algorithm that makes at most $c|L|^{1/k}$ tests in the first round and at most $|M||L|^{1/k}\log|R|/c^{1/(k-1)}$ tests in each subsequent round.*

Proof. Let $t = c|L|^{1/k}$. Partition L into t pieces $L_1, \ldots, L_t$ of size $|L|/t$. In round 1, test $L_i \cup R$ for each i. At most $|M|$ of those sets can contain an edge, say $L_{i_1} \cup R, \ldots, L_{i_m} \cup R$, where $m \leq |M|$. Apply the first bipartite lemma to the matching problems on those m sets. Let e_j denote the number of edges in L_{i_j}. The number of tests performed in round 1 is $t = c|L|^{1/k}$, and in each subsequent round it is at most

$$\sum_j e_j(|L|/t)^{1/(k-1)}\log|R| = |M||L|^{1/k}\log|R|/c^{1/(k-1)} \qquad \square$$

THEOREM 8.6. *For $3 \leq k \leq \log n$, there is a deterministic k-round algorithm for the matching problem that asks $O(n^{1+1/(2(k-1))}(\log n)^{1+1/(k-1)})$ queries per round.*

Proof. After adding $o(n)$ virtual unmatched points we may assume that n is of the form $p^2 + p + 1$, where p is prime; these virtual points will be omitted from any actual tests. In round 1, construct a projective plane with n points, and perform one test for each line. Delete every line that contains no edge of the matching. Construct G and its color classes as in the coloring lemma. If $(x, y) \in M$, then x and y must belong to distinct color classes of G. For each pair of color classes apply the bipartite lemma with $c = \log n^{1/(k-1)}/\log n$ and the number of rounds $= k - 1$. The number

of tests in round 2 is at most

$$O\left(\binom{\sqrt{n}\log n}{2} c\sqrt{n}^{1/(k-1)}\right) = O(n^{1+1/(2(k-1))}(\log n)^{1+1/(k-1)}).$$

The number of tests performed in any of the rounds 3 through k is bounded by

$$O\left((n/2)\sqrt{n}^{1/(k-1)} \log \sqrt{n}/c^{1/(k-2)}\right) = O(n^{1+1/(2(k-1))}(\log n)^{1+1/(k-1)}). \qquad \square$$

Our 2-round deterministic algorithm uses finite projective spaces of dimension 4 in a somewhat different way. It has the advantage of using queries whose size is approximately $n^{1/4}$ or less.

THEOREM 8.7. *There is a deterministic* 2*-round algorithm that asks* $\frac{5}{4}n^{3/2}(1 + o(1))$ *queries of size at most* $n^{1/4}$ *each.*

Proof. Choose $m \approx n^{1/4}$ such that K_n is the disjoint union of approximately $n^{3/2}$ copies of K_m. (Use a projective or affine space of dimension 4 where each line has length m.) (1) Ask one query for each copy of K_m. At most $n/2$ of them can contain an edge. (2) Use brute force to find those edges. $\square$

9. Open problems.

- Determine the smallest possible constant c such that there is a deterministic nonadaptive algorithm for the matching problem on n vertices that makes $c\binom{n}{2}(1 + o(1))$ queries.
- Find more efficient deterministic k-round algorithms or prove lower bounds for the number of queries in such algorithms.

Acknowledgments. We would like to thank Lance Fortnow, László Lovász, and Oded Regev for helpful discussions. The second author gratefully acknowledges a Temple University study leave.

REFERENCES

[1] N. ALON AND J. H. SPENCER, *The Probabilistic Method*, 2nd ed., Wiley, New York, 2000.
[2] R. BEIGEL, N. ALON, M. S. APAYDIN, L. FORTNOW, AND S. KASIF, *An optimal procedure for gap closing in whole genome shotgun sequencing*, in Proceedings 2001 RECOMB, ACM Press, New York, 2001, pp. 22–30.
[3] M. CLAUSTRES, P. KJELLBERG, M. DESGEORGES, H. BELLET, P. SARDA, H. BONNET, AND C. BOILEAU, *Detection of deletions by the amplification of exons (multiplex PCR) in Duchenne muscular dystrophy*, J. Genet. Hum., 37 (1989), pp. 251–257 (in French).
[4] V. GREBINSKI, *On the power of additive combinatorial search model*, in Proceedings of the 4th Annual International Computing and Combinatorics Conference, Lecture Notes in Comput. Sci. 1449, Springer-Verlag, New York, 1998, pp. 194–203.
[5] V. GREBINSKI AND G. KUCHEROV, *Optimal query bounds for reconstructing a Hamiltonian cycle in complete graphs*, in Proceedings of the 5th Israeli Symposium on Theoretical Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 166–173.
[6] V. GREBINSKI AND G. KUCHEROV, *Reconstructing a Hamiltonian cycle by querying the graph: Application to DNA physical mapping*, Discrete Appl. Math., 88 (1998), pp. 147–165.
[7] V. GREBINSKI AND G. KUCHEROV, *Optimal reconstruction of graphs under the additive model*, Algorithmica, 28 (2000), pp. 104–124.
[8] M. HALL, *Combinatorial Theory*, 2nd ed., Wiley, New York, 1986.
[9] H. TETTELIN, D. RADUNE, S. KASIF, H. KHOURI, AND S. SALZBERG, *Pipette optimal multiplexed PCR: Efficiently closing whole genome shotgun sequencing project*, Genomics, 62 (1999), pp. 500–507.
[10] R. M. WILSON, *Decomposition of complete graphs into subgraphs isomorphic to a given graph*, Congr. Numer., 15 (1975), pp. 647–659.

SIAM J. COMPUT.
Vol. 33, No. 2, pp. 502–503

ON RATIONAL NUMBER RECONSTRUCTION AND APPROXIMATION*

VICTOR Y. PAN† AND XINMAO WANG‡

Abstract. The celebrated LKS algorithm by Lehmer, Knuth, and Schönhage, combined with the product tree technique, enables an equivalently rapid alternative to our recent modification of the extended Euclidean algorithm for the reconstruction of a rational number from its modular as well as numerical approximations.

Key words. rational number reconstruction, rational number approximation, extended Euclidean algorithm, LKS algorithm, product tree technique

AMS subject classifications. 68W40, 68W30, 68Q25

DOI. 10.1137/S0097539703437181

Our algorithm in [7] uses $O(M(d)\log d)$ bit-operations to solve the rational number reconstruction problem of recovering a unique rational number x/y from three integers k, v, and $u = (x/y) \bmod v$, where it is assumed that $k, v, |x|$, and y are integers in $2^{O(d)}$, v and y are coprime, $k > 2|x|, v/k > y > 0$, the problem has a solution, and $M(d)$ bit-operations suffice to multiply two integers modulo 2^d. In a dual variation, our algorithm solves the rational number approximation problem; that is, for a given triple of positive integers k, m, and $n > m$, it computes a unique pair of coprime integers p and q such that $0 < q < k + 1$ and $|m/n - p/q| < 1/(2k^2)$. The bit-cost is in $O(M(d)\log d)$ provided k, m, and n are in $2^{O(d)}$ and the problem has a solution. (For more details see [7] and the bibliography therein.)

Now we wish to point out an alternative solution algorithm for both problems which also uses $O(M(d)\log d)$ bit-operations. It relies on the well-known LKS algorithm by Lehmer, Knuth, and Schönhage (see [5], [3], [6]), which uses $O(M(d)\log d)$ bit-operations to compute all partial quotients in the extended Euclidean algorithm applied to two positive integers u and v in $2^{O(d)}$ (the quotients are represented by 2-by-2 matrices P_i in [7], and our algorithm computes only some selected quotients). Having all the quotients available, one may apply the product tree technique [3, Theorem 1] and easily arrive at the desired solutions x/y and p/q to the problems of the rational number reconstruction and approximation, respectively. Namely, one may apply this technique to compute (within the desired time bound of $O(M(d)\log d)$) an appropriate positive integer j and a pair of 2-by-2 matrices Q_j and Q_{j-1} (which are the products of the matrices P_i from 1 to j and $j-1$, respectively) such that the desired ratio x/y is recovered from them immediately, e.g., in the same way as in [7]. Similarly the ratio p/q can be computed. We refer the readers to the paper [1] and the bibliography therein (also see the relevant chapters in the books [4] and [2]) for the detailed technical and historical account on the LKS algorithm and the product tree technique.

*Received by the editors September 5, 2003; accepted for publication October 28, 2003; published electronically March 5, 2004.
http://www.siam.org/journals/sicomp/33-2/43718.html
†Department of Mathematics and Computer Science, Lehman College of CUNY, Bronx, NY 10468 (vpan@lehman.cuny.edu).
‡Ph.D. Program in Mathematics, Graduate School of CUNY, New York, NY 10036 (xwang2@gc.cuny.edu).

Acknowledgments. We thank Dr. Erich Kaltofen, who suggested extending the LKS algorithm to yield fast rational number reconstruction, and the referee for helpful editing advice.

REFERENCES

[1] D. J. Bernstein, *Fast Multiplication and Its Applications*, preprint, 2003; available from http://cr.yp.to/papers.html.
[2] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 2nd ed., Cambridge University Press, Cambridge, UK, 2003.
[3] D. E. Knuth, *The analysis of algorithms*, in Actes du congrès internationale des mathématiciens, tome 3, Gauthier-Villars, Paris, 1971, pp. 269–274.
[4] D. E. Knuth, *The Art of Computer Programming. Volume* 2: *Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1997.
[5] D. H. Lehmer, *Euclid's algorithm for large numbers*, Amer. Math. Monthly, 45 (1938), pp. 227–233.
[6] A. Schönhage, *Schnelle Berechnung von Kettenbruchentwicklugen*, Acta Inform., 1 (1971), pp. 139–144.
[7] X. Wang and V. Y. Pan, *Acceleration of Euclidean algorithm and rational number reconstruction*, SIAM J. Comput., 32 (2003), pp. 548–556.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 505–543

ON UNIVERSAL CLASSES OF EXTREMELY RANDOM CONSTANT-TIME HASH FUNCTIONS*

ALAN SIEGEL†

Abstract. A family of functions F that map $[0, m-1]$ into $[0, n-1]$ is said to be κ-wise independent if any tuple of κ distinct points in $[0, m-1]$ have a corresponding image, for a randomly selected $f \in F$, that is uniformly distributed in $[0, n-1]^\kappa$. This paper shows that for suitably fixed $\epsilon < 1$ and any $\kappa < m^\epsilon$, there are families of κ-wise independent functions that can be evaluated in constant time for the standard random access model of computation. It is also proven that any such family requires a storage array of m^δ random seeds for a suitable $\delta < 1$. These seeds can be pseudorandom values precomputed from an initial $O(\kappa)$ random seeds. A simple adaptation yields n^ϵ-wise independent functions that require n^δ storage in many cases where $m \gg n$. Lower bounds are presented to show that neither storage requirement can be materially reduced. Previous constructions of random functions having constant evaluation time and sublinear storage exhibited only a constant degree of independence. Unfortunately, the explicit randomized constructions, while requiring a constant number of operations, are far too slow for any practical application. However, nonconstructive existence arguments are given, which suggest that this factor might be eliminated. The problem of eliminating this factor is shown to be equivalent to a fundamental open question in graph theory. As a consequence of these constructions, many probabilistic algorithms—from traditional hashing to Ranade's emulation of common PRAM algorithms—can for the first time be shown to achieve, up to constant factors, their expected asymptotic performance for a programmable, albeit formal and currently impractical, model of computation, and a research direction is now available that may eventually lead to implementations that are fast and provably sound.

Key words. hash functions, universal hash functions, hashing, limited independence, optimal speedup, PRAM emulation, storage-time tradeoff

AMS subject classifications. 86R10, 68Q10, 68Q22, 68Q30

DOI. 10.1137/S0097539701386216

1. Introduction.

Hash functions have been a mainstay in the design of many randomized and high performance algorithms, and their use has been especially important for theoretical developments in large-scale parallel and distributed computation. In particular, storage is often hashed to distribute data evenly across memory modules, to balance access patterns to memory, and to balance load requirements across networks. Although some algorithms can achieve optimal performance improvements via precomputed randomization, and others can exploit adaptive methods to achieve maximum performance, many fundamental problems have yet to be solved without the statistically uniform randomness supported by idealized or by (the more pragmatic) universal classes of hash functions. Ranade, for example, uses polynomials of degree $8 \log n$ to randomize routing and memory accesses in his emulation scheme for the common PRAM [20]. To date, his results have yet to be matched without such a hashing procedure.

Of course, good hash functions are also important for sequential computation. For example, it is well known that for uniform hashing, the expected cost to insert the $(\alpha n + 1)$st item into a table of size n is $\frac{1}{1-\alpha} - o(1)$ probes when fully random hash functions are used [12]. Yet the significance of even this performance bound is by no means clear. The difficulty is that the result has been proven for idealized

*Received by the editors February 28, 2001; accepted for publication (in revised form) October 15, 2003; published electronically March 23, 2004. This work was supported in part by ONR grant N00014-85-K-0046 and NSF grants CCR-8906949, CCR-8902221, CCR-9204202, and CCR-9503793.
http://www.siam.org/journals/sicomp/33-3/38621.html
†Courant Institute, New York University, New York, NY 10012-1185 (siegel@cs.nyu.edu).

fully independent hash functions, which cannot be efficiently computed. Suppose, for example, we wish to select a random mapping from $[0, n^2 - 1]$ into $[0, n - 1]$. Since there are n^{n^2} such mappings, it follows that the program length of such a function must be at least $n^2 \log n$ bits, on average, which is much larger than the hash table it is intended to service.

On the other hand, results based on idealized randomness sometimes translate into average case performance guarantees for real computation. For example, double hashing uses the same insertion and retrieval strategies as uniform hashing, but defines the jth probe for a key x to be $(d(x) + (j-1)f(x)) \bmod p$, for prime p and random functions d and f, with ranges $[0, p-1]$ and $[1, p-1]$. This scheme requires about $2\lceil \log p \rceil$ random bits per hash key, and we could take these bits to be any fixed portions of the key itself, provided it is at least $2\lceil \log p \rceil$ bits long. Then a performance analysis for the truly random case can be viewed as an average case argument for this deterministic formulation, where the averaging is over all possible sequences of data keys. However, for uniform hashing—or indeed any hashing scheme where each probe sequence is defined by more bits than are available in a hash key—even this weak interpretation would seem to lack adequate justification.

Accordingly, it is to notions of computable randomness that we now turn our attention.

1.1. Background and overview. Carter and Wegman introduced *universal classes of hash functions* [5] to provide a theoretically sound and pragmatic framework for programmable hash functions with limited degrees of freedom. Subsequent developments in universal hashing (cf. [36, 18]) have used comparable formulations which are all basically as follows.

DEFINITION 1.1. *A family of hash functions F with domain D and range R is* (κ, μ)-wise independent *if it is finite and for all $y_1, y_2, \ldots, y_\kappa \in R$, for all* distinct *$x_1, x_2, \ldots, x_\kappa \in D$,*

$$|\{f \in F : f(x_i) = y_i,\ i = 1, 2, \ldots, \kappa\}| \leq \mu \frac{|F|}{|R|^\kappa}. \tag{1.1}$$

Consequently, if a function $f \in F$ is chosen at random with all elements equally likely to be selected, then f will map any fixed set of κ distinct points from D into R^κ according to a probability distribution that is, in some sense, roughly uniform.

In these definitions, D and R are always finite. It is worth pointing out that any function is a hash function, and from the perspective of universal hashing, any function is a bad hash function. What matters are the statistical characteristics of the family members as quantified in (1.1). (Of course, we are also concerned with the program size and operation count associated with evaluating the functions in such a family.) It is important to notice that (κ, μ)-wise independence implies (j, μ)-wise independence for $j < \kappa$. Indeed, if we sum both sides of (1.1) over all $y_\kappa \in R$, the constraint on $f(x_\kappa)$ becomes the trivial $f(x_\kappa) \in R$, and the bound reverts to the exact requirement for $(\kappa - 1, \mu)$-wise independence.

For expositional simplicity, we will, unless stated otherwise, set $\mu = 1$ and simply refer to (κ)-wise independence. Likewise, we will use *limited independence* to refer to families where the independence parameter κ is much smaller than $|D|$, and take *fully independent* to mean that $\kappa = |D|$.

Carter and Wegman exhibited the following families of (κ, μ)-wise independent

hash functions where $D = [0, p-1]$, $R = [0, n-1]$, and p is prime:

$$F_{(\kappa)} = \left\{ f \mid f(x) = \left(\sum_{j=0}^{\kappa-1} a_j x^j \bmod p \right) \bmod n,\ a_j \in [0, p-1] \right\}. \tag{1.2}$$

They showed (among other results) that these functions give a performance for hashing with separate chaining that is effectively indistinguishable from that for fully random functions. In this hashing scheme, the elements of a set $\aleph \subseteq [0, p-1]$ are mapped by a randomly selected function $f \in F_{(\kappa)}$ (chosen independent of $\aleph$) into linked lists that are accessed via an array $B[0, n-1]$. When multiple items are mapped to a common address location j, the items are chained together in a list with the head list item stored in $B[j]$. The remaining list members are kept in auxiliary storage outside of B. Carter and Wegman used the fact that the sum of the expected jth moments of the list lengths, when fully random hash functions are used, is essentially the same as that resulting from the use of random functions in $F_{(\kappa)}$, provided $j \leq \kappa$. For hashing with separate chaining, the expected average of the second moments of the list lengths determines the expected retrieval time, whence pairwise independence guarantees an expected performance that is equivalent to that resulting from fully independent hash functions.

Subsequently, Mehlhorn and Vishkin presented an extensive collection of different hash functions that meet the requirements of (κ, μ)-wise independence [18]. Yet despite a surprising diversity of definitions and algebraic structures, every one of their (κ)-wise independent functions—like all such families ever discovered—requires at least κ arithmetic operations per evaluation. Under the circumstances, it is natural to wonder if this is a coincidence. We ask:

Is there an inherent κ-time penalty for computing
(κ)-wise independent hash functions, or can we do better?

The answer will turn out to be that faster functions are indeed programmable. However, a complete solution is fairly technical and cannot be stated in terms of the standard formulations for limited randomness. The definitions must be extended to expose additional statistical characteristics that affect the computational resources in cases where the independence exceeds the number of random seeds used to hash an individual key.

Families of random hash functions are often defined as a three-tier mapping $g \circ f \circ h$, where $h : D \to D_1$, $f : D_1 \to D_1$, and $g : D_1 \to R$. The underlying domain D might be huge, in which case a preliminary mapping h is used to map the hash keys into a smaller domain that is better suited for efficient computation. Thus, D_1 might be a finite field with elements that can be represented by, say, a finite number of machine words, and the actual target range R might be a table index that is less well suited for defining families of uniformly distributed hash functions. For example, the Carter–Wegman class (1.2) actually defines a $(\kappa, 1)$-wise independent family $f \in F$ on the field of integers mod$p$, and then uses $g(x) \equiv x \bmod n$ to project the hashing onto the intended range $[0, n-1]$. Consequently, the resulting composition fails to achieve $(\kappa, 1)$-wise independence because the first $p \bmod n$ points in $[0, n-1]$ have $\lceil \frac{p}{n} \rceil$ preimage points, whereas the remaining $n - (p \bmod n)$ points have $\lfloor \frac{p}{n} \rfloor$ preimage points. This mild nonuniformity in the postprocessing was the reason for introducing the parameter μ in the definition of (κ, μ)-wise independence. However, for the problems we consider, the pathologies associated with g are insignificant, whereas the irregularities associated with h turn out to have a greater effect on the underlying computational resources.

Formally, we analyze the computational costs for computing (κ)-wise independent mappings from a domain D into a range R. The underlying parameters are 1) the number σ of initializing random seeds, 2) the number of operations $T < \kappa$ that must be executed per evaluation of the hash function, 3) the size of the hashing program, and 4) the size of a provably necessary auxiliary storage array M, which, it will turn out, we can presume to hold pseudorandom numbers that have been precomputed from σ initial seeds. Additional parameters are the domain and range sizes $|D|$ and $|R|$. This paper gives a lower bound to show that M must store at least $|D|^\epsilon$ words from R for suitable fixed $\epsilon < 1$. We also show how to circumvent this lower bound. By allowing an asymptotically negligible chance (which is yet another parameter) that the hash function family will fail to achieve (κ)-wise independence for a given data set, the lower bound will still apply, but for an effective domain size $|D_1|$, which might be far smaller than $|D|$.

The exact details include additional parameters, but they play a minor role in the resource requirements. In all cases, the requisite number of random seeds remains modest: $\sigma = \Theta(\kappa)$.[1]

We present two kinds of algorithmic formulations for such high performance hash functions. In particular, nonconstructive existence arguments are used to show that algorithms might, in principle, match the resource constraints predicted by the lower bound. Our (nonconstructive) hash functions require a precomputation phase where the σ random seeds are used to precompute $|D|^\epsilon$ pseudorandom values that are stored in an auxiliary array M. Then a suitable algorithm can compute highly random hash values by using its hash key deterministically to select a small number T of elements from M and by simply returning the sum of these elements $\bmod |R|$, or, say, their bitwise exclusive-or. The computation can be formulated to use, say, just twice as many elements from M as are proved necessary by our bound. Moreover, this number T turns out to be constant: $T = \Theta(1)$. On the other hand, the question of which T words should be selected from M lies at the heart of the nonconstructive existence argument.

Although the question of how to select these T random words effectively is still open, we show that hash functions with very high independence are indeed programmable, provided we accept the less satisfactory evaluation time of $T^{\Theta(T)}$, where T is the $O(1)$ operation count predicted by nonconstructive methods.

1.2. Related work. A preliminary version of this work appeared as an extended abstract in [26]. The current presentation includes more efficient existential formulations, better probabilistic constructions, a stronger and more general lower bound, and an improvement in the applications. In the interim, several works have appeared that use the properties of these high performance hash functions [9, 11, 16].

The theory of hashing has also progressed along several independent lines. In particular, Dietzfelbinger et al. [7] pursued the perfect hashing methods begun by Fredman, Komlós, and Szemerédi [8] to develop hash functions that can locate data without collisions. The processing is fully dynamic, with an average insertion time of $O(1)$ steps per insertion, and a resulting 1 table probe per data request. While these techniques have many useful properties, and might comprise the methods of choice for many applications, they do not construct functions with the statistical randomness that is the primary objective of this work. Further, the functions require auxiliary

[1] For very large domains, the prehashing in the appendix, which is based on [8, 17], uses an additional $\log\log|D|$ bits, which are provably necessary (cf. [8, 17]). On the other hand, the convention that words can be processed in unit time is suspect for domains where such a term would matter.

storage of $\Omega(n)$ bits, which is in excess of the storage we will use.

Many works have used universal classes of hash functions to reduce the resource conflicts that arise in distributed computation. For example, Karp, Luby, and auf der Heide use the highly random constant-time hash functions described in the preliminary version of this work to resolve various problems of contention at the memory module level [11]. Their model is designed to avoid some of the idealized characteristics of PRAMs, where contention occurs only at the level of memory cells. Other such uses include developments by Goldberg, Matias, and Rao [9] and Matias and Schuster [16]. On the other hand, some machine contention can be resolved with much simpler hash functions. For example, Kruskal, Rudolph, and Snir use such an approach to emulate an $(n^{1+\epsilon})$-processor PRAM on an n-processor machine [13].

1.3. The organization. This paper is organized as follows. Section 1 concludes with a brief description of the computational model and an introduction to the calculus of limited independence. Section 2 analyzes three random function formulations, which range from the nonconstructively defined but extremely efficient to the programmable (with code). They all run in constant time and can be as much as $(|D|^{\delta})$-wise independent for different, suitably small constants $\delta > 0$.

Section 3 presents lower bounds for the resource requirements of fast hash functions. Section 4 discusses the intrinsic resource tradeoffs for the general hashing formulations of section 2, their lower bounds, and their equivalence to a fundamental problem in graph theory. Section 5 gives a few applications and section 6 presents the conclusions.

1.4. Computational assumptions. The performance results are based on a variety of assumptions that characteristically hold for most hashing problems, as well as some decisions to use problem descriptors aimed at keeping the parameters at hand as simple as possible. The first point of clarification is that the main problem size parameter, in all hashing problems, will be n. For example, we will typically have $\theta(n)$ hash keys that must be mapped into some range. Similarly, some applications feature n processors that at a given time step each hash one memory address. With pipelining, the n processors might each have as many as $\log n$ addresses to hash. We state this explicit dependence on n because n will sometimes be only implicitly represented by, say, a domain size parameter m, which might be set to n^{ℓ}, for suitable ℓ. Second, we assume that for some constant $r \geq 2$, the family of hash functions we construct will only need to be (κ, μ)-wise independent with a probability exceeding $1 - n^{-r}$, so that a negligible percentage of the hash functions might be seriously defective.

It is easy to adapt the results to larger data sets, but to avoid clutter we will use these adaptations only when necessary. In addition, we will either suppose that the domain size $|D|$ is polynomial in n, or assume that there is "no computation charge" for premapping elements from a very large space D into, say, $[0, n^{\ell} - 1]$ for some constant $\ell \geq r + 2$. It will suffice to use a pairwise independent hash function (i.e., a randomly selected member of a universal class of pairwise independent hash functions) for the premapping $h : D \to [0, n^{\ell} - 1]$.

We will typically suppose that the range R is no larger than n^{γ} for some constant γ. Otherwise, the presumption that simple arithmetic operations take constant time is again somewhat suspect. It is also the case that for much larger number ranges, the size of the initializing random seeds will have to grow. Our constructions and lower bounds can accommodate these changes, and the total operation count for the hash functions will remain constant. We will sometimes assume that $n \leq |R|$. This assumption is harmless; if $|R|$ is much less than n, it will suffice to use a range $[0, \hat{R}-1]$,

where $\hat{R}$ is a multiple of $|R|$, and postprocess the hash functions mod$|R|$.

The underlying computational model (once the domain is reduced in size) is the standard Random Access Machine.

Last, we assume that there is a source of truly independent uniformly distributed random seeds in any interval $[0, \rho-1]$, and that modest quantities of them are available for use. In particular, if a hash function is to be (κ)-wise independent, then $\Theta(\kappa)$ random seeds can be supplied as initializing input to the hashing algorithm. Of course, if strings of $\theta(\kappa)$ seeds have relative probabilities of occurrence that are not equal but instead differ by as much as a factor of μ, then the resulting hash functions will have statistics that exhibit the same multiplicative variability.

1.5. A primer on limited independence and the simplest derived inequalities. The probabilistic inequalities used in this paper all follow from a few closely related formulations. Let $\mathcal{X}(*) : (T, F) \to (1, 0)$ be the indicator function, which maps Boolean expressions into 1 when they are true and 0 otherwise. Let g be a nonnegative increasing function, and let X be a random variable. Then $\frac{g(X)}{g(a)}$ is at least 1 whenever $X > a$, and is nonnegative everywhere. Thus $\text{Prob}\{X > a\} = \text{Prob}\{g(X) > g(a)\} = \text{E}[\mathcal{X}(\frac{g(X)}{g(a)} > 1)] \le \text{E}[\frac{g(X)}{g(a)}] = \frac{\text{E}[g(X)]}{g(a)}$. Several functions g are commonly used for proving probabilistic inequalities of the form

$$\text{Prob}\{X > a\} \le \frac{\text{E}[g(x)]}{g(a)}. \tag{1.3}$$

When X is itself nonnegative, setting $g(x) = x$ gives what is sometimes referred to as Markov's inequality.

This scheme also works if g is an increasing multinomial or multivariate function that is never negative on its domain. Let, for example, the random variable Φ be the sum of n Bernoulli trials: $\Phi = z_1 + z_2 + \cdots + z_n$, where z_k is 1 if, say, the hash function f maps x_k to memory module 1, and 0 otherwise. In this case, z_k is itself the disjoint sum of many atomic events: $z_k = \bigvee_j (f(x_k) = j)$, where the j index ranges over all addresses in module 1. If f is selected from a (κ, μ)-wise independent family, then, as is straightforward to verify, the Bernoulli trials will be (κ, μ)-wise independent. A suitable multinomial g would be $g(z_1, z_2, \ldots, z_n) = \binom{\Phi}{\kappa}$, which has a very natural interpretation. In this case, $\binom{\Phi}{\kappa}$ is the sum of all products of subsets of κ distinct z's: $\binom{\Phi}{\kappa} = \sum_{1 \le i_1 < i_2 < \cdots < i_\kappa \le n} z_{i_1} z_{i_2} \cdots z_{i_\kappa}$.

We recall that the expectation and multiplication commute for independent random variables: $\text{E}[YX] = \text{E}[X]\text{E}[Y]$ if X and Y are independent. In the case of (κ, μ)-wise independence, $\text{E}_\kappa[X_1 X_2 \cdots X_\kappa] \le \mu \prod_{1 \le j \le \kappa} \text{E}_\infty[X_j]$, for nonnegative (κ, μ)-wise independent random variables $X_1, X_2, \ldots, X_\kappa$, where E_κ denotes the expectation under (κ, μ)-wise independence, and E_∞ denotes the expectation under full randomness. More generally, we note that if the X_i are (κ, μ)-independent, then so are $\{g_i(X_i)\}_{j=1}^{\kappa}$ for any set of functions g_i. (These facts can be verified by formulating events as the logical-or of κ-way atomic events that assign each X_i fixed values.) Under (κ, μ)-wise independence, it follows that

$$\text{Prob}\{\Phi > a\} \le \frac{\text{E}_\kappa[\binom{\Phi}{\kappa}]}{\binom{a}{\kappa}} \le \frac{\mu}{\binom{a}{\kappa}} \sum_{1 \le i_1 < i_2 < \cdots < i_\kappa \le n} \text{E}_\infty[z_{i_1}]\text{E}_\infty[z_{i_2}] \cdots \text{E}_\infty[z_{i_\kappa}].$$

The bound is useless if $a < \kappa$, since the denominator would be zero; in such a case, we might replace κ with a smaller value.

In many of our applications, $\mathrm{E}_\infty[z_k] = p$ for some expression p that is independent of k. In these cases, $\mathrm{E}_\infty[\binom{\Phi}{\kappa}] = \binom{n}{\kappa}p^\kappa$, and hence $\mathrm{Prob}\{\Phi > a\} \le \mu\frac{\binom{n}{\kappa}p^\kappa}{\binom{a}{\kappa}}$.

These formulations can also be used to bound tail expectations as well as probabilities. For example, if X is integer valued and nonnegative, then

$$\mathrm{E}[X \cdot \mathcal{X}(X \ge a)] \le k\mathrm{E}\left[\frac{\binom{X}{k}}{\binom{a-1}{k-1}}\right]$$

for $k \le a$. Similarly,

$$\mathrm{E}[X] \le a + \mathrm{E}[X \cdot \mathcal{X}(X \ge a)]$$

for $a \ge 0$, and likewise

$$\mathrm{E}[X^2] \le a^2 + k(k-1)\mathrm{E}\left[\frac{\binom{X+1}{k}}{\binom{a-1}{k-2}}\right],$$

provided that X is nonnegative and integer valued. If $X_1, X_2, \ldots, X_n$ are random variables, then

$$\mathrm{E}[\max_i X_i] \le a + \sum_i \mathrm{E}[X_i \cdot \mathcal{X}(X_i \ge a)],$$

and if the X_i are identically distributed, then

$$\mathrm{E}[\max_i X_i] \le a + n\mathrm{E}[X_1 \cdot \mathcal{X}(X_1 \ge a)].$$

These formulations suggest that we will need some basic combinatorial inequalities. Some use the notation $n^{\underline{k}} \equiv n(n-1)\cdots(n-k+1)$. For example, when $m > n$, then we can write $\frac{n^{\underline{\kappa}}}{m^{\underline{\kappa}}} < (\frac{n}{m})^\kappa$, and $\frac{m^{\underline{\kappa}}}{n^{\underline{\kappa}}} > (\frac{m}{n})^\kappa$ without appealing to any estimate for factorials.

A fairly elementary form of Stirling's formula says that $n! > n^{n+1/2}e^{-n}$. Sometimes even more naive formulations such as $n! > n^n e^{-n}$ will suffice. For example, the last inequality implies that $n^{\underline{k}} > n^k e^{-k}$. (Indeed, to see that the former implies the latter, observe that $n^{\underline{\kappa}} > \frac{\kappa^\kappa}{e^\kappa \kappa!}n^{\underline{\kappa}} = \frac{\kappa^\kappa}{e^\kappa} \cdot \frac{n^{\underline{\kappa}}}{\kappa^{\underline{\kappa}}} > (\frac{\kappa}{e})^\kappa \cdot (\frac{n}{\kappa})^\kappa = (\frac{n}{e})^\kappa$.) It now follows that

$$\frac{n^k e^{-k}}{k!} \le \binom{n}{k} \le \frac{n^k}{k!},$$

and we will bound many combinatorial expressions by replacing $\binom{n}{k}$ by $\frac{n^k}{k!}$, if the expression appears in a numerator, and by $\frac{n^k e^{-k}}{k!}$, if it is in a denominator. As an immediate application, we have

$$\mu\frac{\binom{n}{\kappa}p^\kappa}{\binom{a}{\kappa}} \le \mu\frac{(np)^\kappa}{a^{\underline{\kappa}}} \le \mu\left(\frac{npe}{a}\right)^\kappa,$$

which further simplifies the preceding estimates for $\mathrm{Prob}\{\Phi > a\}$, when Φ is the sum of n (κ, μ)-wise independent Bernoulli trials with probability of success p. Although much better estimates can be derived, they will not be needed.

Many powerful inequalities come from setting $g(X) = e^{\lambda X}$, in (1.3), and then solving for the value of $\lambda > 0$ that gives the best bound. If X is the sum of n mutually independent identically distributed random variables z_i, then $\mathrm{E}[e^{\lambda X}] = \mathrm{E}[e^{\lambda z_1}]^n$, and $\mathrm{Prob}\{X > a\} < e^{-\lambda a}\mathrm{E}[e^{\lambda z_1}]^n$. If $X = z_1 + z_2 + \cdots + z_\kappa$, where $\{z_i\}_{i=1}^{\kappa}$ are random variables that enjoy (κ, μ)-wise independence, then $\mathrm{E}_\kappa[e^{\lambda X}] \le \mu \prod_{i=1}^{\kappa} \mathrm{E}_\infty[e^{\lambda z_i}]$. A more complete set of these estimates for cases with limited independence can be found in [24].

Sometimes an event $\mathcal{E}$ will have a corresponding predicate $\mathrm{Pred}(*, *)$ and sets U and W that satisfy the following: if $\mathcal{E}$ occurs, then there exist $u \subset U$ and $w \subset W$ such that $\mathrm{Pred}(u, w)$ is true. This implies that $\mathrm{Prob}\{\mathcal{E}\} \le \sum_{u \subset U,\, w \subset W} \mathrm{E}[\mathcal{X}(\mathrm{Pred}(u, w))]$, where $\mathcal{X}$ is the 0–1 indicator function. We will take care to ensure that these expectations can be evaluated (or estimated) under the restrictions of (κ, μ)-wise independence. For example, a predicate $\mathcal{E}$ might have a decomposition as a disjoint sum of atomic κ-way events:

$$\mathcal{E} = \bigvee_{(y_1, y_2, \ldots, y_\kappa) \in \Omega} \left(\bigwedge_{1 \le i \le \kappa} f(x_i) = y_i \right)$$

for some set Ω of κ-tuples. Any such $\mathcal{E}$ has a probability that is, up to the factor μ, bounded by the analogous probability that results from full independence and the uniform distribution.

Armed with this set of counting arguments, we proceed to the problem of building fast computable hash functions.

2. The hash functions. The first step is to formalize a somewhat stronger and occasionally more applicable notion of restricted randomness.

DEFINITION 2.1. *A family of hash functions F with domain D and range R is* r-practical (κ, μ)-wise independent *if for any subset $D_0 \subset D$, with $|D_0| \le \sqrt{|R|^r}/2$, there is a subset of functions $\widehat{F} \subseteq F$ where $|\widehat{F}| \ge |F|(1 - 2\frac{\binom{|D_0|}{2}}{|R|^r})$ and for all $y_1, y_2, \ldots, y_\kappa \in R$, for all* distinct *$x_1, x_2, \ldots, x_\kappa \in D_0$,*

$$\frac{|\widehat{F}|}{\mu |R|^\kappa} \le |\{f \in \widehat{F} : h(x_i) = y_i,\ i = 1, 2, \ldots, \kappa\}| \le \frac{\mu |\widehat{F}|}{|R|^\kappa}. \tag{2.1}$$

We define F to be uniformly r-practical (κ, μ)-wise independent *if* (2.1) *holds for some $|\widehat{F}| \ge |F|(1 - \frac{1}{|R|^r})$ with $D_0 = D$.*

This definition differs from those in the literature in two respects. First, the r-practical part of the formulation has been introduced to formalize the notion that a negligible fraction of the hash functions might be defective for the specific data that is to be processed. The reason for weakening the definition is the following. For very large domains, a prehashing function h should be used to map the data into a smaller domain D_1. But then a collection of n keys from D will, with a probability of (about) $\binom{n}{2}/|D_1|$, have a collision under the projection h. Consequently, under the best of circumstances, the composition of h with a well-designed hash function on D_1 will only be able to achieve high degrees of independence with a probability of $1 - \binom{n}{2}/|D_1|$, since once two keys are mapped to the same value, any subsequent processing will preserve this equality. We will typically set $|D_1| \approx n^4$. With this understanding, we can focus on the design of hash functions defined on domains of reasonable size, and just revisit the issue of highly disparate domain sizes when establishing the corresponding lower bounds.

In addition to accommodating small percentages of faulty hash functions, this definition differs from the usual formulations by specifying two-sided constraints. This formulation (which becomes an equality when $\mu = 1$) is intended to support inclusion-exclusion arguments based on (κ)-wise independent hash functions. (In [22], for example, such a two-sided bound is essential.) As a practical matter, this additional constraint is not particularly burdensome. While most constructions of (κ, μ)-wise independent function classes have focused on one-sided constraints as typified by Definition 1.1, actual constructions, such as the polynomials $F_{(\kappa)}$ of (1.2), usually satisfy the more restrictive standard[2] of Definition 2.1.

2.1. Existence arguments. We now examine the problem of constructing families of (κ)-wise independent hash functions that map a domain $D = [0, m-1]$ into a range $R = [0, \rho - 1]$. The basic motivation for the approach comes from considering the following question. Imagine, for the moment, that $D = R$, and is of prime size. Then we can use κ random elements from D as coefficients in (1.2) to construct a (κ)-wise independent hash function. Evidently, evaluation requires $O(\kappa)$ time. If m random elements are provided, then table lookup gives an $O(1)$ time (m)-wise independent hash function. What sort of random functions can be constructed from m^ϵ random seeds?

For the purposes of this section, we will require that the underlying storage be sufficient to hold the random function code, including its random seeds. Our model of computation is the Random Access Machine, where both memory access and the basic arithmetic and logic operations can be executed on words in unit time (cf. [1, Chap. 1]). Each word will comprise $\lceil \log_2 \rho \rceil$ or possibly $\lceil \log_2 m \rceil$ bits.

We temporarily suppress the issue of program size and prove the existence of families of fast highly independent hash functions that map $[0, m-1]$ into $[0, \rho - 1]$ and use m^ϵ words of random input. We will also initially ignore all preprocessing and postprocessing steps as well as any concern associated with huge domains to study the problem of constructing fully $(\kappa, 1)$-wise independent hash functions that map $D = [0, m-1]$ into a suitable $R = [0, \rho - 1]$, given an auxiliary array of m^ϵ random words from R, for some[3] $\epsilon < 1$. Evidently, any random hash function must have a mechanism to associate each element in D with a few of these random words, as otherwise no random computation can result. If the association is not adaptive,[4] then it can be represented by a bipartite graph G on the vertex sets D and $D^\epsilon \equiv [0, m^\epsilon - 1]$. Such a bipartite graph must associate at least l random numbers with each set of l elements from D, for $l \leq \kappa$, as otherwise there are not enough degrees of freedom to achieve (κ)-wise independence. Suitable graphs are formalized as follows.

DEFINITION 2.2. *Let an* (m, ϵ, d, κ) local concentrator *be a bipartite graph on sets of vertices* I *(inputs) and* O *(outputs), where* $|I| = m$, $|O| = m^\epsilon$, *and the following hold. Each input has outdegree* d. *Every set of* j *inputs, for* $j \leq \kappa$, *has edges to some set of* j *or more outputs.*

Our next observation is that such graphs exist, even with very small outdegree d. From this narrow perspective, the parameter r is extraneous in the following lemma

[2]For some applications, the y_i of Definition 2.1 might be required to be distinct. Alternatively, the functions of (1.2) might include the additional term x^κ.

[3]We will avoid the clutter of floors and ceilings as long as possible, and assume that the exponent ϵ has been chosen to make the resulting expression such as m^ϵ an integer.

[4]The lower bound will include adaptive probing, where the location of the next random key to retrieve can depend on the value of other such data that has already been read. We discuss nonadaptive approaches because they are more intuitive, and explain all of our constructions, which turn out to be nonadaptive.

because the existence result holds for $r = 0$. However, by increasing r from 0 some positive quantity, a suitably structured random graph will be as described with a probability exceeding $1-m^{-rd}$, which can be with overwhelming likelihood as opposed to a probability exceeding zero. The same issue applies to the r in Lemma 2.9.

LEMMA 2.3. *Let d, κ, m, and m^ϵ be positive integers with $\epsilon < 1$. Suppose that $r \geq 0$ and $d \geq \frac{1+\epsilon+r}{\epsilon - \frac{\log \kappa}{\log m}}$. Let $G = (V, E)$ be a random bipartite graph with input vertices I, and output vertices O, where $|I| = m$ and $|O| = m^\epsilon$, and where each vertex in I has edges to d distinct randomly selected neighbors in O. Then with probability exceeding $1 - m^{-rd}$, G is an (m, ϵ, d, κ) local concentrator.*

Proof. We use standard probabilistic arguments (cf. [29]) to estimate the probability that G fails to have j outputs for some set of j inputs for $j \leq \kappa$. By construction, failure cannot occur for $j \leq d$. For larger aggregates of size at most κ, the probability of a failure is bounded by the expected number of pairs $(u \subset I, w \subset O)$, where $|u| = j$, $|w| = j-1$, and all jd edges from u have destinations within w for $d < j \leq \kappa$. Evidently, the probability that the jd edges are so selected, for any fixed (u, w), is

$$\left(\frac{\binom{j-1}{d}}{\binom{m^\epsilon}{d}}\right)^j < \left(\frac{j-1}{m^\epsilon}\right)^{jd}$$

(where the inequality is strict because $d > 1$). The number of candidate (u, w) pairs is just

$$\binom{m}{j}\binom{m^\epsilon}{j-1}.$$

Thus

$$\begin{aligned}
\text{Prob\{failure\}} &< \sum_{d<j\leq\kappa} \binom{m}{j}\binom{m^\epsilon}{j-1}\left(\frac{j-1}{m^\epsilon}\right)^{jd} \\
&< \sum_{d<j\leq\kappa} \frac{m^j m^{j\epsilon-\epsilon} j^{jd}}{j!(j-1)!m^{\epsilon jd}} \\
&< m^{-\epsilon} \sum_{d<j\leq\kappa} \frac{1}{j!(j-1)!}\frac{m^{j+j\epsilon}\kappa^{jd}}{m^{\epsilon jd}}.
\end{aligned}$$

Now, the constraint for d can be written as $m^{d\epsilon} \geq \kappa^d m^{1+\epsilon+r}$, whence $m^{-r} \geq \frac{\kappa^d m^{1+\epsilon}}{m^{d\epsilon}}$. Substituting gives

$$\begin{aligned}
\text{Prob\{failure\}} &< m^{-\epsilon} \sum_{d<j\leq\kappa} \frac{m^{-rj}}{j!(j-1)!} \\
&< m^{-rd}. \qquad \square
\end{aligned}$$

Setting $r = 0$, we see that a randomly constructed graph fails to be an (m, ϵ, d, κ) local concentrator with a probability that is less than 1. It follows that such a construction will succeed with positive probability, and hence these graphs do indeed exist.

We have, as yet, no hash function, but each element, at least, is now associated with a few random values. The obvious use for these values is as coefficients of a hashing polynomial. By increasing the number of random values used in this calculation, we can turn a local concentrator into a calculation procedure for $(\kappa, 1)$-wise independent hash functions.

Let G be an (m, ϵ, d, κ) local concentrator. Let p be prime. For each input i in G, let i's d neighbors in G be stored in the set $\mathrm{Adj}(i)$. Let M_w be an $m^\epsilon \times d$ array of words in $[0, p-1]$, whose concatenated contents are the very long word $w \in [0, p-1]^{m^\epsilon d}$.

Define the random hash function

$$f_w^G(i) = \left(\sum_{0 \le k < d} \sum_{j \in \mathrm{Adj}(i)} i^k M_w(j,k) \right) \bmod p.$$

Thus, computing $f_w^G(i)$ requires $d^2 - 1$ additions and $d-1$ multiplications plus a comparable number of modular divisions. The result turns out to be (κ)-wise independent when the concatenated contents of the storage array M_w are equally likely to be any value in $[0, p-1]^{dm^\epsilon}$.

THEOREM 2.4. *Let G be an (m, ϵ, d, κ) local concentrator. Then $\{f_w^G\}_{w \in [0, p-1]^{dm^\epsilon}}$ is a $(\kappa, 1)$-wise independent family of hash functions mapping $[0, m-1]$ into $[0, p-1]$.*

Proof. Let $x_1, x_2, \ldots, x_\kappa$ be any κ distinct values in $[0, m-1]$, and $y_1, y_2, \ldots, y_\kappa$ be κ arbitrary values in $[0, p-1]$. We must show that for all x and y assignments, there are exactly the same number of functions f_w that satisfy the system

$$f_w^G(x_i) = y_i \bmod p \quad \text{for } i = 1, 2, \ldots, \kappa. \tag{2.2}$$

Now the unknowns, in this system, are the word assignments to the array M_w, and the system is linear in these variables. The equations in (2.2) comprise κ constraints in dm^ϵ unknowns. So if the system enjoys linear independence, then the set of κ equations in (2.2) will have exactly $p^{dm^\epsilon - \kappa}$ different w words that are solutions, which ensures the precise uniformity required of (κ)-wise independence. It follows that we need only establish the linear independence of all systems defined in (2.2).

We prove the linear independence by contradiction. In the context of Lemma 2.3, $[0, m-1]$ plays the role of I, and $[0, m^\epsilon - 1]$ is O. The linear system $f_w^G(x_i) = y_i$ for $i = 0, 1, 2, \ldots, m-1$ has the explicit formulation

$$\sum_{k=0}^{d-1} (\mathcal{N})^k \times \mathcal{A} \times (M_{j,k})_{j \in O} = (y_0, y_1, \ldots, y_{m-1})^T \bmod p, \tag{2.3}$$

where $\mathcal{N}^k$ is the diagonal matrix with i^k in location (i, i), for $i = 0, 1, \ldots, m-1$ (with $\mathcal{N}^0$ set to the identity matrix), and $\mathcal{A} = (a_{i,j})_{i \in I,\, j \in O}$ is the adjacency matrix for G, which has m rows, m^ϵ columns, and d 1's in each row. The representation $(M_{j,k})_{j \in O}$ is intended to denote the kth of d column vectors, which each have $|O| = m^\epsilon$ rows. Taken together, they comprise the entries in the auxiliary array M of random words.

The proof will exploit the structural characteristic of G to establish the linear independence of any subsystem of $|I_0| \le \kappa$ equations for $I_0 \subset [0, m-1]$. Such subsystems can be written as follows:

$$\sum_{0 \le k < d} \sum_{j \in \mathrm{Adj}(i)} i^k M_w(j,k) = y_i \bmod p \quad \text{for } i \in I_0.$$

Let I_0 be a set of row indices in a subsystem of (2.3) that is linearly dependent and has no proper subset of equations that is linearly dependent. Suppose $|I_0| \le k$.

By definition, I_0, in the local concentrator, has $d|I_0|$ edges, which reach at least $|I_0|$ outputs. Let O_0 comprise the vertices adjacent to I_0 in G. Then the average number of edges received from I_0 by a vertex in O_0 is $d|I_0|/|O_0|$ which is at least 1 and at

most d, since $|I_0| \le O_0 \le d|I_0|$. By the pigeonhole principle, there must be an output $o \in O_0$ having exactly q edges that originate in I_0 for some q, where $1 \le q \le d$. Let I_1 be this set of neighbors of o in I_0: $I_1 = \{i \in I_0 : o \in \mathrm{Adj}(i)\}$. Let $I_1 = \{i_1, i_2, \ldots, i_q\}$, where $1 \le q \le d$. Now consider the linear subsystem with rows indexed by I_1 and columns restricted to the variables $M(o,k)$, where $k = 0, 1, \ldots, d-1$. This system comprises the following submatrix of a Vandermonde matrix:

$$\begin{pmatrix} 1 & i_1 & i_1^2 & \ldots & i_1^{d-1} \\ 1 & i_2 & i_2^2 & \ldots & i_2^{d-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & i_q & i_q^2 & \ldots & i_q^{d-1} \end{pmatrix}.$$

As is well known, such a subsystem cannot be linearly dependent since no two rows are the same, and $q \le d$. Since none of the rows with indices in $I_0 - I_1$ has any of the variables $M(o,0), M(o,1), \ldots, M(o,d-1)$ present (that is, they are present with coefficients of zero), the only way a linear combination of the rows can add to the zero vector is if each row in I_1 has a coefficient of zero. Thus, the assumption that the system is dependent and minimal is contradicted. □

So far, we have a probabilistic fast hashing procedure that is (κ)-wise independent, uses dm^ϵ random words of $\log p$ bits, and requires $\Theta(d^2)$ operations. The construction gives a generic transformation from a graph rich in matchings to a family of highly random functions. We now give a more efficient construction that makes better use of random graph properties. The graph will require an outdegree d that is slightly larger than before, but only one random value will be stored in each output destination. The construction uses a sparse bipartite graph where every set of κ rows of its adjacency matrix is linearly independent, and this independence even extends to computations over finite commutative groups such as the integers $\bmod \rho$ for any integer ρ.

Definition 2.5. *Let G be a bipartite graph on sets of vertices I (inputs) and O (outputs), where $|I| = m$, $|O| = m^\epsilon$. Let* $\mathrm{Adj}(i)$*, for $i \in I$, be the set of i's neighbors in O. We say that G is* (m, ϵ, d, κ) locally peelable *if each input node has an outdegree of at most d and the following holds: for any set I_0 of κ or fewer input vertices, some node in I_0 has a neighbor that is not a neighbor of any other node in I_0. Formally, for all $I_0 \subset I$, where $|I_0| \le \kappa$, there is an $i_0 \in I_0$ such that*

$$\mathrm{Adj}(i_0) - \bigcup_{i \in I_0 - \{i_0\}} \mathrm{Adj}(i) \neq \emptyset.$$

Lemma 2.6. *Let $D = [0, m-1]$, and $R = [0, \rho-1]$. Let $\bigoplus$ denote addition* $\bmod \rho$ *or any other commutative group operator that is closed over R. Let G be (m, ϵ, d, κ) locally peelable, with $\epsilon < 1$. For each input i in G, let i's neighbors in G be stored in the set* $\mathrm{Adj}(i)$*. Let M_w be an array of m^ϵ words from R, where the concatenated content of M_w comprises the very long word $w \in [0, \rho-1]^{m^\epsilon}$.*

Define the hash function

$$f_w^G(i) = \bigoplus_{j \in \mathrm{Adj}(i)} M_w(j) \quad \text{for } i \in D.$$

Then $\{f_w^G\}_{w \in [0, \rho-1]^{m^\epsilon}}$ is a $(\kappa, 1)$-wise independent family of hash functions mapping D into R.

Proof. Consider the subsystem (in unknowns w) $f_w^G(x_i) = y_i$, for $i = 1, 2, \ldots, \kappa$, which assigns κ values in R to κ distinct input vertices in D. Since the system is

locally peelable, it can be permuted into a system where the first κ columns comprise an upper triangular matrix with 1's along its diagonal. The first row corresponds to an input x_{i_0} reaching an output o_0 that is not a neighbor of any node in $\{x_i\}_{i=1}^{\kappa} - \{x_{i_0}\}$. The columns are permuted so that the first column corresponds to o_0. Then the same reordering is recursively applied to the remaining $\kappa - 1$ equations in unknowns $O - \{o_0\}$. Given such an upper triangular permutation of the system, there are $m^\epsilon - \kappa$ column variables that are not among the columns that contain the κ diagonal entries, and we are free to assign arbitrary values from R to these variables. Then the remaining κ variables will have a unique solution that can be found by back-solving via the $\bigoplus$ operator. Thus, the number of solutions to any κ such equations in the m^ϵ unknowns is exactly $\rho^{m^\epsilon - \kappa}$, which ensures that the f_w^G are $(\kappa, 1)$-wise independent. □

We now show that some random graphs are locally peelable.

DEFINITION 2.7. *Let an* (m, ϵ, d, κ) local expander *be a bipartite graph on sets of vertices I (inputs) and O (outputs), where $|I| = m$ and $|O| = m^\epsilon$, and the following hold. Each input has a positive outdegree bounded by d. Any set of j inputs, for $1 < j \le \kappa$, has edges to at least $\lfloor jd/2 \rfloor + 1$ different outputs.*

LEMMA 2.8. *An* (m, ϵ, d, κ) local expander *is* (m, ϵ, d, κ) *locally peelable.*

Proof. The proof is by contradiction. Suppose that I_0 is a smallest set of input variables that is not locally peelable. Obviously $|I_0| > 1$. Suppose $|I_0| \le \kappa$. By definition, the input variables of I_0 have at least $\lfloor jd/2 \rfloor + 1$ different output variables, which have, on average, at most $jd/(\lfloor jd/2 \rfloor + 1) < 2$ different input neighbors in I_0. By the pigeonhole principle, some output vertex o_0 must therefore have just one input neighbor i_0 in I_0. We peel off this input, and observe that the subgraph with $|I_0| - 1$ input variables is locally peelable, whence the supposition that $|I_o| \le \kappa$ must be false. □

As with Lemma 2.3, the following existence argument includes the parameter r to increase the likelihood of success from something positive to a probability overwhelmingly close to 1.

LEMMA 2.9. *Let d, κ, m, and m^ϵ be positive integers with $\epsilon < 1$. Suppose that $r \ge 0$ and $\epsilon > \frac{2+r}{d} + \frac{1+\ln \frac{d}{2} + \ln \kappa}{\ln m}$. Let $G = (V, E)$ be a random bipartite graph with input vertices I and output vertices O, where $|I| = m$ and $|O| = m^\epsilon$, and where each vertex in I has edges to d distinct randomly selected neighbors in O. Then with probability exceeding $1 - m^{-r}$, G is an (m, ϵ, d, κ) local expander.*

Proof. Rewriting the bound for ϵ gives $m^{\epsilon d/2} > m(\kappa de/2)^{d/2} m^{r/2}$, so that

$$\frac{m(jde/2)^{d/2}}{m^{\epsilon d/2}} < m^{-r/2} \quad \text{for } j \le \kappa. \tag{2.4}$$

Proceeding as in Lemma 2.3 gives the following estimates for the probability that G is not a local expander:

$$\begin{aligned} \text{Prob\{failure\}} &< \sum_{1<j\le\kappa} \binom{m}{j} \binom{m^\epsilon}{\lfloor jd/2 \rfloor} \left(\frac{\lfloor jd/2 \rfloor}{m^\epsilon} \right)^{jd} \\ &< \sum_{1<j\le\kappa} \frac{m^j m^{\epsilon jd/2} (\lfloor jd/2 \rfloor)^{jd}}{j! (\lfloor jd/2 \rfloor)! m^{\epsilon jd}}, \end{aligned}$$

whence approximating $(\lfloor jd/2 \rfloor)!$ via our two Stirling estimates (depending on the

parity of jd) and simplifying give

$$\text{Prob}\{\text{failure}\} < \sum_{2\le j\le\kappa} \left(\frac{m(jde/2)^{d/2}}{m^{\epsilon d/2}}\right)^j \frac{1}{j!}.$$

Applying (2.4) gives

$$\text{Prob}\{\text{failure}\} < \sum_{2\le j\le\kappa} m^{-rj/2}/j! < m^{-r}\sum_{2\le j}\frac{1}{j!} < m^{-r}. \qquad \square$$

Setting $r = 0$ shows that such a graph exists. Of course, actual use of this lemma will require that ϵ be less than 1, which requires that $d \ge 3$.

Combining Lemmas 2.6, 2.8, and 2.9 gives the following.

THEOREM 2.10. *Let d, κ, m, and m^ϵ be positive integers with $\epsilon < 1$. Suppose that $1 > \epsilon \ge \frac{2}{d} + \frac{1+\ln d+\ln\kappa}{\ln m}$. Let G be an (m,ϵ,d,κ) local expander. Let $\bigoplus$ define a commutative group over $[0,\rho-1]$. Let*

$$f_w^G(i) = \bigoplus_{j\in \text{Adj}(i)} M_w(j).$$

Then $\{f_w^G\}_{w\in[0,\rho-1]^{m^\epsilon}}$ is a $(\kappa,1)$-wise independent family of hash functions mapping $[0,m-1]$ into $[0,\rho-1]$.

Proof. Lemma 2.9 ensures that G exists, when $\epsilon \ge \frac{2}{d} + \frac{1+\ln d+\ln\kappa}{\ln m}$. Lemma 2.8 ensures that G is (m,ϵ,d,κ) locally peelable. Lemma 2.6 shows that $\{f_w^G\}_{w\in[0,\rho-1]^{m^\epsilon}}$ is $(\kappa,1)$-wise independent. $\square$

In particular, $\bigoplus$ can be addition modρ, or $\bigoplus$ can be the bitwise exclusive-or operator, where ρ is power of 2 and the integers in $[0,\rho-1]$ are regarded as binary strings of length $\log\rho$.

It is worth observing that the m^ϵ-word seeds w as used in Theorem 2.10 can be produced by random members of a universal class H of $(d\kappa,\mu)$-wise independent hash functions.

COROLLARY 2.11. *Let H be a universal class of $(d\kappa,\mu)$-wise independent hash functions that map $[0,m^\epsilon-1]$ into $[0,\rho-1]$. For $h\in H$, let w_h be the concatenation of $h(0),h(1),\ldots,h(m^\epsilon-1)$. Then the family $\{f_{w_h}^G\}_{h\in H}$ is (κ,μ)-wise independent.*

Proof. Observe that each set of range assignments to a κ-tuple of elements in D induces a (locally peelable) linear system of κ equations in at most $d\kappa$ index locations (i.e., variables) of the auxiliary array M. We can extend the set to include exactly $d\kappa$ such variables. Then the solution set can be represented by $\rho^{d\kappa-\kappa}$ different $d\kappa$-tuple assignments for this collection. So if each assignment occurs with probability $\rho^{-d\kappa}$, then the resulting functions will be $(\kappa,1)$-wise independent by Theorem 2.10. If each tuple of $d\kappa$ assignments has a probability of occurrence that is reweighted by a factor of $\mu\in[\mu_1,\mu_2]$, then the resulting aggregate probability that F computes the prespecified κ-tuple of range assignments is likewise reweighted by a factor that is within this interval. $\square$

Consequently, the space-time tradeoff, for families of fast highly independent hash functions, is not a function of the number of random seeds that must be specified (which is just $\Theta(\kappa)$) but is really a matter of intrinsic storage requirements for the auxiliary storage array M.

Although the second construction gives a more efficient family of hash functions and also provides a generic procedure that turns a good graph into a family of hash

functions, it does not quite supersede the first construction because there are no known explicit graphs of either type. Should a short (deterministic or effective probabilistic) algorithm be found to define local concentrators where an input's adjacency list can be generated from a small number of instructions, then fast, highly independent hash functions will follow. Similarly, effective procedures for constructing local expanders will yield even better hash functions.

2.2. Asymptotically compact constructions and their randomization. From a positive perspective, we have proven that one good graph is all we need: the contents of the auxiliary random seed array M defines the different members in the associated family of hash functions. Furthermore, we have shown that good graphs not only exist, but can even be formulated so that randomized constructions can build them with very high likelihood.

Unfortunately, no deterministically defined graph has been proven, as yet, to satisfy either expander-like formulation. We now show that from a theoretical perspective where impractical constants might be tolerated, there are asymptotically spatially compact programmable formulations of constant-time (n^ϵ)-wise independent hash functions. In particular, we will show that Cartesian products can be used to obtain compact representations of (wildly) less efficient hash functions, where we forgo some randomness and increase the $O(1)$ operation count by an exponentially larger constant, but reduce the overall storage requirements to $O(n^\epsilon)$ for suitable $\epsilon < 1$. These variations can be applied to either of our nonconstructive formulations, but we will focus on those built from locally peelable graphs, since they appear to be more efficient.

The first step is to show that such compact representations exist. The second is to observe that simple changes in the random constructions yield locally peelable graphs with a high probability, so that the graph itself can be part of the random seed specification. The result will be a uniform r-practical family where a good graph need not be found. Instead, the initializing random seeds will include values to define G. As a consequence, an asymptotically negligible fraction of these implicitly defined graphs will fail to have the necessary expander-like properties.

DEFINITION 2.12. *Let the* Cartesian product $G \bigotimes H$ *of two bipartite graphs* $G = (I, O; E)$ *and* $H = (J, Q; F)$ *be the graph* $\mathcal{G} = (\mathcal{I}, \mathcal{O}; \mathcal{E})$*, with input vertex set* $\mathcal{I} = I \times J$*, output set* $\mathcal{O} = O \times Q$*, and edge set* $\mathcal{E}$*, which contains the edge from* $(i, j) \in \mathcal{I}$ *to* $(o, q) \in \mathcal{O}$ *if and only if* $\text{edge}(i, o) \in E$ *and* $\text{edge}(j, q) \in F$.

LEMMA 2.13. *Let* $G = (I, O; E)$ *be* (m, ϵ, d, κ) *locally peelable, and* $H = (J, Q; F)$ *be* (n, ϵ, c, κ) *locally peelable. Then the Cartesian product* $G \bigotimes H$ *is* $(mn, \epsilon, cd, \kappa)$ *locally peelable.*

Proof. We need only verify the local peelability property for $G \bigotimes H$. The proof is by contradiction. Let X be a smallest set of input variables for $G \bigotimes H$ that is not locally peelable. Obviously $|X| > 1$. Suppose $|X| \leq \kappa$. Let $I_G = \{i \mid \exists j : (i, j) \in X\}$. Now $|I_G| \leq \kappa$, and hence there is an $\hat{\imath} \in I_G$ with a neighbor o_1 that is not a neighbor of any node in $I_G - \{\hat{\imath}\}$. Now let $J_G = \{j \mid \exists j : (\hat{\imath}, j) \in X\}$, and let $\ell = |J_G|$. By definition, J_G is locally peelable in H. So let $q_1, q_2, \ldots, q_\ell$ be a sequence of outputs peeled one-by-one in order from J_G. It is easy to see that $(o_1, q_1), (o_1, q_2), \ldots, (o_1, q_\ell)$ now gives a comparable peeling for $\{o_1 \times J_G\}$ in X. But then the remaining $|X| - \ell$ variables (if any) are by assumption locally peelable, whence the supposition that there is a subgraph with at most κ input vertices that is not locally peelable must be false. ∎

Consequently, if G is an $(m^\epsilon, \epsilon_1, d, \kappa)$ local expander, then applying Lemma 2.13 a

total of $c = \lceil \frac{1}{\epsilon_1} \rceil - 1$ times shows that the Cartesian product $\bigotimes_{j=1}^{\lceil 1/\epsilon \rceil} G$ is $(m^{c\epsilon}, \epsilon_1, d^c, \kappa)$ locally peelable.

Lemma 2.13 plus the constructions of Theorem 2.10 establishes the following.

THEOREM 2.14. *Let the positive integers m, m^ϵ, κ, d, and ρ be specified with $m > \rho$, and suppose that $\frac{1}{2} \geq \frac{\epsilon}{2} \geq \frac{2}{d} + \frac{1+\ln d + \ln \kappa}{\epsilon \ln m}$, and $m^{\epsilon^2/2} > \frac{1}{\epsilon}$. Let $c = \lceil \frac{1}{\epsilon} \rceil$ and set $\epsilon_1 = \frac{\log \lceil m^{\epsilon/c} \rceil}{\epsilon \log m}$. Let $\mathcal{G}$ be the set of all graphs $G = (V, E)$ that are bipartite with m^ϵ input vertices and $m^{\epsilon_1 \epsilon}$ output vertices, and where each input vertex has edges to d distinct neighbors among the outputs. Then the following holds.*

There is a $G \in \mathcal{G}$ that is $(m^{c\epsilon}, \epsilon_1, d^c, \kappa)$ locally peelable.

Let $\widehat{G} = \bigotimes_{i=1}^{c} G$. The set $\{f_w^{\widehat{G}}\}_{w \in [0, \rho-1]^{m^{\epsilon_1 \epsilon}}}$ is a universal class of $(\kappa, 1)$-wise independent hash functions that map $[0, m-1]$ into $[0, \rho - 1]$.

The program for $f_w^{\widehat{G}}$ comprises $O(\epsilon^2 d) m^\epsilon \log m$ bits and computes a hash value by retrieving no more than $d^{1+1/\epsilon}$ words from an auxiliary storage array of $O(m^\epsilon)$ random words in $[0, \rho - 1]$.

Proof. Lemma 2.9 establishes the existence of a suitable G. Programmatically, G will be stored and used as part of the hash function. A value $i \in [0, m-1]$ is hashed by computing the adjacency list for i in the implicitly defined $\widehat{G} \equiv \bigotimes_{j=1}^{c} G$. Given the local peelability of G, Lemma 2.13 shows that $\widehat{G}$ is $(m^{c\epsilon}, \epsilon_1, d^c, \kappa)$ locally peelable, which is adequate for use in Theorem 2.10, since $c\epsilon > 1$.

It is not difficult to verify that all parameter sizes are as stated. In particular, there are three issues: the local peelability of G, the size of the auxiliary array, and the outdegree of each vertex in $\widehat{G}$.

Parameter ϵ_1 satisfies $\frac{1}{c} \leq \epsilon_1$. By construction, $c < \frac{1}{\epsilon} + 1$, whence $\frac{1}{c} > \frac{\epsilon}{1+\epsilon} > \frac{\epsilon}{2}$. Hence $\frac{\epsilon}{2} < \frac{\epsilon}{1+\epsilon} < \epsilon_1$, so ϵ_1 satisfies the local peelability inequality of Lemma 2.9 since $\frac{\epsilon}{2}$ does.

By definition,

$$\epsilon_1 = \frac{\ln \lceil m^{\epsilon/c} \rceil}{\epsilon \ln m} < \frac{\ln(m^{\epsilon/c} + 1)}{\epsilon \ln m} = \frac{1}{c} + \frac{\ln(1 + m^{-\epsilon/c})}{\epsilon \ln m} < \frac{1}{c} + \frac{m^{-\epsilon/c}}{\epsilon \ln m} \leq \frac{1}{c} + \frac{m^{-\epsilon^2/2}}{\epsilon \ln m}.$$

Now, $m^{\epsilon^2/2} \geq \frac{1}{\epsilon}$, so $\epsilon_1 - \frac{1}{c} < \frac{1}{\ln m}$. Consequently, the word count for the auxiliary array is

$$m^{\epsilon \epsilon_1 c} = m^{\epsilon(\epsilon_1 - 1/c + 1/c)c} \leq m^\epsilon m^{\frac{\epsilon c}{\ln m}} = m^\epsilon (m^{\frac{1}{\ln m}})^{c\epsilon} = m^\epsilon e^{\epsilon c} < m^\epsilon e^2.$$

The outdegree is $d^c < d^{\frac{1}{\epsilon}+1}$.

Thus, the hash function has three parts. One part comprises the array M of $m^{c\epsilon\epsilon_1} = O(m^\epsilon)$ random words from $[0, p-1]$, which requires $O(m^\epsilon \log p)$ bits. Another stores the graph G, which requires m^ϵ strings of d words that are each $O(\log(m^{\epsilon^2}))$ bits long. The requisite storage for G (i.e., part 2) is, therefore, $O(\epsilon^2 d m^\epsilon \log m)$ bits. The third part of the function is the finite program that uses G and M to evaluate the hash function for an element in $[0, m-1]$. ∎

Remarks. Theorem 2.14 can be used with m replaced by n^k, ϵ replaced by ϵ/k, κ replaced by n^δ, and d unchanged. The inequality for ϵ then becomes

$$\frac{\epsilon}{2k} \geq \frac{2}{d} + \frac{1 + \ln d + \delta \ln n}{\epsilon \ln n}, \tag{2.5}$$

which we can write as $\epsilon \geq \frac{4k}{d} + 2k \frac{1+\ln d + \delta \ln n}{\epsilon \ln n}$. Upon applying the simplification $\epsilon > a + \sqrt{b}$ implies $\epsilon \geq a + b/\epsilon$, we can conclude that the requirement for ϵ is met if

$\epsilon \geq \frac{4k}{d} + \sqrt{2k(\delta + \frac{1+\ln d}{\ln n})}$. The corresponding hash functions can be evaluated with $O(d^{k/\epsilon})$ operations, and have a program size of $O(n^\epsilon)$ words. It is worth pointing out that (2.5) can permit ϵ to be any sufficiently small positive constant provided, say, $d > 12k/\epsilon$, $\delta < \epsilon/6$, $n > d^{6/\epsilon}$, and $n^{k\epsilon^2/2} > \frac{1}{\epsilon}$. The resulting functions are $(n^{\epsilon/6}, 1)$-wise independent.

For completeness, we state without proof the analogous compaction/expansion formulation for local concentrators.

LEMMA 2.15. *Let G be an $(m^\epsilon, \epsilon, d, \kappa)$ local concentrator. Then the Cartesian product $\bigotimes_{j=1}^{1/\epsilon} G$ is an $(m, \epsilon, d^{1/\epsilon}, \kappa)$ local concentrator.*

So far, families of hash functions mapping D into R have been "demonstrated" only in a probabilistic sense; no explicit constructions have been given. However, by increasing, slightly, the degrees of freedom in our probabilistic constructions, the same counting arguments ensure that with probability $1 - \frac{1}{|R|^r}$, a randomly selected graph is a local concentrator or expander. Consequently, we can include, in the initial seeding, enough random data to build a local expander as well as the array of random words it will be used to access. The graph will not be prohibitively large if it defines a compact version as formulated in Theorem 2.14 and adapted in (2.5). The only necessary accommodation is to keep the probability of failure appropriately tiny in the rescaled parameters, which can be done by increasing r by a factor of $\frac{1}{\epsilon}$. The resulting randomized construction $F_M^{G(m,\epsilon,d)}$ is an explicit family of $O(1)$ time hash functions that is uniformly r-practical (κ)-wise independent as characterized by Definition 2.1. Last, it is worth pointing out that the graph structure suggests that the number of initializing random seeds need not change by more than a constant factor; it is sufficient to build G from pseudorandom numbers generated from a traditional $(\kappa d^{k/\epsilon})$-wise independent hash function. To see that this is so, it suffices to observe that the proof of Lemma 2.9 was based on expectations of disjunctions of atomic events comprising the conjunction of $d\kappa$ random edge assignments.

These observations are formalized in the following theorem, where the bipartite graph G is just a collection of $(\kappa d^{k/\epsilon})$-wise independent pseudorandom numbers. A simple algorithm to process this graph and compute the actual hash function it encodes is presented after Theorem 2.16. The construction is randomized and asymptotically compact.

THEOREM 2.16. *Let the positive integers n, k, n^ϵ, κ, d, ρ, and r be specified, and suppose that $\frac{\epsilon}{k+1} \geq \frac{2}{d} + \frac{r}{d\epsilon} + \frac{1+\ln d+\ln\kappa}{\epsilon \ln n}$. Let $c = \lceil \frac{k}{\epsilon} \rceil$ and set $\epsilon_1 = \frac{\log\lceil n^{\epsilon/c}\rceil}{\epsilon \log n}$. Suppose that $\epsilon < \frac{k}{k+1}$ and $n^{\epsilon^2/(k+1)} > \frac{1}{\epsilon}$. Let $\mathcal{G}(d, \kappa d^{kc})$ be a set of pseudorandomly generated bipartite graphs with n^ϵ input vertices and $n^{\epsilon\epsilon_1}$ output vertices and where each vertex has edges to d distinct randomly selected neighbors, and let these d-tuple assignments of neighbors be (κd^{kc-1})-wise independent. Let M be a collection of long words that are the concatenation of (κd^{kc})-wise independent sequences of $n^{\epsilon\epsilon_1 c}$ words that individually belong to $[0, \rho - 1]$. For any $G \in \mathcal{G}(d, \kappa d^{kc})$, let $\widehat{G} = \bigotimes_{i=1}^{ck} G$. Given $w \in M$ and $G \in \mathcal{G}$, let f_w^G be the hash function defined by $f_w^{\widehat{G}}$ as in Lemma 2.6. Let $F_M^{\mathcal{G}} = \{f_w^G\}_{w \in M,\, G \in \mathcal{G}}$.*

Then $F_M^{\mathcal{G}}$ is an explicit family of uniformly r-practical $(\kappa, 1)$-wise independent hash functions that map $[0, n^k - 1]$ into $[0, \rho - 1]$. Furthermore, each $w \in M$ comprises $|w| = n^{\epsilon\epsilon_1 c} \leq e^{k+1} n^\epsilon$ words from $[0, \rho - 1]$. The long words in M can be generated from κd^{kc} random seeds that belong to $[0, \rho - 1]$. Each $G \in \mathcal{G}$ can be stored as a sequence of n^ϵ words of $O(\frac{\epsilon^2}{k} \log n)$ bits, and can be generated from (κd^{kc}) random seeds that

are evenly distributed among the ranges $[0, n^{\epsilon\epsilon_1} - 1], [0, n^{\epsilon\epsilon_1} - 2], \ldots, [0, n^{\epsilon\epsilon_1} - d]$. *The probability that a specific* $F_*^{\widehat{G}}$ *fails to be* $(\kappa, 1)$*-wise independent is bounded by* n^{-r}.

Proof. The arguments are just a combination of the reasoning given in Lemma 2.9, the rescaling listed in (2.5), and, mutatis mutandis, the calculations given in the proof of Theorem 2.14.

However, a few comments should be made about how to generate the two tables of pseudorandom data. One potential difficulty is that we need, at initialization time, a small (traditional) hash function to expand some count σ of truly random seeds into a much larger collection of n^δ pseudorandom seeds belonging to $[0, \rho - 1]$, even when $[0, \rho - 1]$ cannot define a finite field. Perhaps the mathematically cleanest solution is to factor ρ into its product of powers of distinct primes, and use a (κd^{ck})-wise independent hash function over each of these fields. Then the Chinese remainder theorem can be used to reconstruct the comparably random $(\kappa d^{kc}, 1)$-wise independent pseudorandom seeds as needed.

Last, we note that straightforward encoding suffices to transform tuples of random values into distinct random values. □

The first requirement for ϵ is satisfied if $\epsilon \geq \frac{2(k+1)}{d} + \sqrt{(k+1)(\frac{r}{d} + \frac{1+\ln d + \ln \kappa}{\ln n})}$.

For applications of Theorem 2.16, a new hash function family (which might have to be found in case $F_*^{\widehat{G}}$ fails to be $(\kappa, 1)$-wise independent) would include new seeds to select a new pseudorandom graph $G \in \mathcal{G}$ as well as new pseudorandom seeds to select a new $w \in M$.

For completeness, this subsection closes with a rather crudely transparent iterative version of the hashing algorithm. The treatment of $\frac{1}{\epsilon}$ and related algebraic expressions as if they were integers is for conceptual transparency and lack of clutter.

function Hash(i: in $[0, n^k - 1]$): in $[0, n^k - 1]$;
  Global M: array of n^ϵ words in $[0, \rho - 1]$;
  Global G: $n^\epsilon \times d$ array of words in $[0, n^{\epsilon^2/k} - 1]$;
  Local $l_1, l_2, \ldots, l_{k/\epsilon}$: in $[0, d - 1]$;
  Local $i_1, i_2, \ldots, i_{k/\epsilon}$: in $[0, n^\epsilon - 1]$;
  Local j: in $[0, n^\epsilon - 1]$;
  Local val: in $[0, n^k - 1]$;
  $val \leftarrow 0$;
  $(i_1, i_2, \ldots, i_{k/\epsilon}) \leftarrow i$; {Distribute i as $\frac{k}{\epsilon}$ different packets of $\epsilon \log n$ bits.}
  for $l_1 \leftarrow 0$ **to** $d - 1$ **do**
    for $l_2 \leftarrow 0$ **to** $d - 1$ **do**
      ⋮
      for $l_{k/\epsilon} \leftarrow 0$ **to** $d - 1$ **do**
        $j \leftarrow (G[i_1, l_1], G[i_2, l_2], \ldots, G[i_{k/\epsilon}, l_{k/\epsilon}])$;
        $val \leftarrow (M[j] \bigoplus val)$
  endallfors; {Altogether, $d^{k/\epsilon}$ $\bigoplus$'s take place.}
return(val).

3. Lower bounds. We now show that the size of our random word array cannot be materially reduced without affecting the running time of the hash function. A family of (κ, μ)-wise independent hash functions $F_M = \{f_m(x)\}_{m \in M}$, where $f_m : D \rightarrow R$, will be modeled as follows. Each f_m is defined by the same algorithm, which inputs x and then reads d locations in an array $A[1..z]$ that contains z values belonging to R. The index m is a very long word comprising the concatenated data contained in A. We can suppose that each computation of f examines exactly d

entries in A. The randomness in the system comes from the choice of input seed strings $m \in M \subseteq R^z$. The set M need not contain all possible sequences of z words from R and can be a multiset, which means that some strings could have several copies present in M. More generally, each $m \in M$ could be assigned a positive real weight, and an element in M could be selected with a probability equal to its fraction of the total weight in M. Although our proof extends directly to cover this case, we will simply assume that all strings are equally likely to be selected. This avoids unnecessary clutter by keeping the lower bound restricted to a more manageable six parameters.

The algorithm can even be viewed as adaptive since we allow a key x to be hashed by a scheme that, for $j = 0, 1, \ldots, d-1$, uses x and the first j values found in A to determine which A location to read for the $(j+1)$st value. In any case, x and the retrieved array values are then used deterministically to compute the random function value in R.

The forthcoming lower bound argument will exploit the following independence constraint. Let $X = \{x_1, x_2, \ldots, x_\kappa\}$ be a set of κ keys in D, and let, for $j = 1, 2, \ldots, \kappa$, the sequence of seed values read to compute $f_M(x_j)$ be $s_{j,1}, s_{j,2}, \ldots, s_{j,d}$. Then the hashing of X, over all possible seed strings in M, cannot cause the same fixed sequence of κd seed values to be read with a probability that exceeds $\frac{\mu}{|R|^\kappa}$, since X would otherwise be mapped into the same κ-tuple with a probability that is too high. Since the determination of the sequence of seed locations can be adaptive, we are obliged to formulate the proof with respect to the triples $(\zeta, M_i^\zeta, D(\zeta, i))$, where ζ is a subset of $\kappa - 1$ locations in the seed array A, i is a string that can result from concatenating the seed values in ζ, M_i^ζ is the subset of seed data sets $m \in M$ for which the concatenation of the seed values on ζ is i, and $D(\zeta, i)$ is the subset of keys in D that are hashed by seeds read solely from ζ, when the seed data is restricted to M_i^ζ. Given these triples, the proof proceeds by summing their respective weights and bounding the respective sums. Bounds are attained by selecting the "good" triples, where $|M_i^\zeta|$ is large enough to ensure that $|D(\zeta, i)| < \kappa$. The lower bound quantifies the intuition that if most $|D(\zeta, i)|$ are small, then A must be big.

THEOREM 3.1. *Let $F_M = \{f_m\}_{m \in M}$ denote a family of (κ, μ)-wise independent hash functions mapping D into R, where $M \subseteq R^z$. Then the number of probes T that must be used to evaluate $f \in F_M$ satisfies either $T \geq \kappa$ or*

$$z^{\frac{T}{}} > (\kappa - 2)^{\frac{T-1}{}} |D| \left(1 - \frac{\mu}{|R|}\right).$$

Proof. Let every evaluation of f examine exactly d entries in the array A. We show that d satisfies the constraint for T. We can also suppose that $\mu < |R|$ as otherwise there is nothing to prove.

For each set ζ of $\kappa - 1$ locations in the z-element array A, let the locations in ζ be sorted by the value of their indices in A, so that each ζ has a fixed sequencing for the values stored in its locations. Then any string $m \in M$ will have, on its restriction to ζ, $\kappa - 1$ values in a fixed order. We can view the concatenation of this subsequence as a number in $[0, |R|^{\kappa-1} - 1]$, so that any such ζ defines an implicit projection of any $m \in M$ into $[0, |R|^{\kappa-1} - 1]$. Given ζ, let M be partitioned into $M^\zeta = \langle M_1^\zeta, M_2^\zeta, \ldots, M_{|R|^{\kappa-1}}^\zeta \rangle$, where M_i^ζ is the set of strings in M that have projections equal, on ζ, to the value $i \in [0, |R|^{\kappa-1} - 1]$. Let $D(\zeta, i)$ be the set of domain elements $x \in D$ that, when computing $f_m(x)$ for $m \in M_i^\zeta$, have their d A-locations read from

within ζ. Let

$$D_0(\zeta,i)=\begin{cases}D(\zeta,i), & \text{provided } |M_i^\zeta|>\mu|M|/|R|^\kappa,\\ \emptyset & \text{if } |M_i^\zeta|\le\mu|M|/|R|^\kappa.\end{cases} \tag{3.1}$$

Given an $x\in D(\zeta,i)$, $f_m(x)$ will be computed by probing the same d-tuple of locations within ζ for all $m\in M_i^\zeta$. It follows that $|D_0(\zeta,i)|<\kappa$ since otherwise there are κ elements in D that hash to some κ-tuple of values in R with a probability that exceeds $\mu/|R|^\kappa$.

There are $\binom{z}{\kappa-1}$ subsets ζ, and each subset induces a partition of M indexed by the m-values restricted to ζ. Let $\Sigma=\sum_\zeta\sum_i|M_i^\zeta||D(\zeta,i)|$, and likewise define the slightly more restricted form $\Sigma_0=\sum_\zeta\sum_i|M_i^\zeta||D_0(\zeta,i)|$. Since $|D_0(\zeta,i)|<\kappa$, $\Sigma_0\le\sum_\zeta\sum_i(\kappa-1)|M_i^\zeta|=\sum_\zeta(\kappa-1)|M|$, whence

$$\Sigma_0\le(\kappa-1)\binom{z}{\kappa-1}|M|. \tag{3.2}$$

On the other hand, Σ has an alternative description, which gives a precise combinatorial formulation. Let $D^+(\zeta,i)$ be the set of pairs (m,x), with $x\in D$ and $m\in M$, such that, when computing $f_m(x)$ for $m\in M_i^\zeta$, all d probes to A-locations lie within ζ. By definition, $|D^+(\zeta,i)|=|M_i^\zeta||D(\zeta,i)|$. Furthermore, since the probing for x uses exactly d locations, each pair (m,x) is counted in exactly $\binom{z-d}{\kappa-1-d}$ different $D^+(\zeta,i)$. Hence

$$\Sigma=\binom{z-d}{\kappa-1-d}|D||M|. \tag{3.3}$$

Finally, the hashing of an $x\in D$ uses d probes that retrieve d specific values in a fixed order from d locations in A. There are at most $|R|^d$ different sequences that can be retrieved, and each sequence corresponds to d distinct locations (since the program need not reread a location when hashing an individual key). Consequently, for a given key x, each possible probe data tuple in R^d could correspond to a specific sequence of d probe locations in A, which can belong to at most $\binom{z-d}{\kappa-1-d}$ different ζ sets, which would have $\kappa-1-d$ unprobed locations that could have up to $|R|^{\kappa-1-d}$ different assignments. Thus, any $x\in D$ belongs to at most $|R|^d\binom{z-d}{\kappa-1-d}|R|^{\kappa-1-d}$ different $D(\zeta,i)$. Hence

$$\Sigma_\zeta\Sigma_i|D(\zeta,i)|\le|D|\binom{z-d}{\kappa-1-d}|R|^{\kappa-1}. \tag{3.4}$$

From (3.1) and the definitions for Σ and Σ_0, we can count that

$$\Sigma-\Sigma_0\le\Sigma_\zeta\Sigma_i\frac{\mu|M|}{|R|^\kappa}|D(\zeta,i)|=\frac{\mu|M|}{|R|^\kappa}\Sigma_\zeta\Sigma_i|D(\zeta,i)|. \tag{3.5}$$

Applying (3.4) to (3.5) shows that $\Sigma-\Sigma_0\le\frac{\mu|M|}{|R|^\kappa}|D|\binom{z-d}{\kappa-1-d}|R|^{\kappa-1}$, whence

$$\Sigma-\Sigma_0\le\binom{z-d}{\kappa-1-d}\mu|M|\frac{|D|}{|R|}. \tag{3.6}$$

Combining (3.3) and inequality (3.6) gives

$$\Sigma_0 \geq \binom{z-d}{\kappa-1-d}|M||D|\left(1-\frac{\mu}{|R|}\right). \tag{3.7}$$

We remark that (3.7) cannot hold as an equality. Indeed, suppose that it did. Then (3.6) and the first inequality in (3.5) would hold as equalities. But if (3.5) were an equality, then $D_0(\zeta,i)$ would have to be the empty set for all i and ζ, since the definitions of $D(\zeta,i)$ and $D_0(\zeta,i)$ ensure that $|M_i^\zeta|(|D(\zeta,i)|-|D_0(\zeta,i)|) \leq \frac{\mu|M|}{|R|^k}|D(\zeta,i)|$, and equality can occur only if $|D_0(\zeta,i)|=0$. So if (3.5) is an equality, then Σ_0 must be zero, which contradicts (3.7), since $\mu<|R|$, $d<\kappa$, and $z\geq\kappa$. Hence

$$\Sigma_0 > \binom{z-d}{\kappa-1-d}|M||D|\left(1-\frac{\mu}{|R|}\right). \tag{3.8}$$

Combining inequalities (3.2) and (3.8) gives

$$(\kappa-1)\binom{z}{\kappa-1}|M| > \binom{z-d}{\kappa-1-d}|M||D|\left(1-\frac{\mu}{|R|}\right).$$

Eliminating common factors establishes that

$$\frac{z^{\underline{d}}}{(\kappa-2)^{\underline{d-1}}} > |D|\left(1-\frac{\mu}{|R|}\right). \qquad \square \tag{3.9}$$

The derivation of (3.9) is valid only if $d<\kappa$, but it is nevertheless reassuring to observe that if we (unjustifiably) set d to κ in the inequality $z^{\underline{d}} > (\kappa-2)^{\underline{d-1}}|D|(1-\frac{\mu}{|R|})$, then the resulting expression collapses to the requirement $z^{\underline{\kappa}}>0$, which is to say that $z\geq\kappa$. Of course κ random numbers are necessary and sufficient, in this trivial case.

There are more significant conclusions to be drawn from Theorem 3.1 and its relationship to Theorem 2.10. These observations are deferred to section 4. Meanwhile, we close this section with two immediate corollaries.

The counting argument for Theorem 3.1 also gives an average case time bound.

COROLLARY 3.2. *Let z, D, and R be fixed. Let $F_M=\{f_m\}_{m\in M}$ denote a family of (κ,μ)-wise independent hash functions mapping D into R, where $M\subseteq R^z$ is the collection of z-word auxiliary array data used to evaluate functions in F_M. Suppose that $\mu\leq|R|/2$. Let T be a lower bound for the worst-case count of the number of probes to the array data that must be used to evaluate some function in any such F_M. Let $\overline{T}$ be the average probe count for any such family, which is averaged over all of D and all of F_M.*

Then $\overline{T}\geq T-\frac{2}{|D|}(\frac{z^{\underline{T-1}}}{(\kappa-2)^{\underline{T-2}}}+\frac{z^{\underline{T-2}}}{(\kappa-2)^{\underline{T-3}}}+\cdots+z)$, which can be expressed as $\overline{T}\geq T-o(1)$ when $z>c\kappa$ for fixed $c>1$.

Proof. The proof is generic. Let Δ_h be the size of the largest domain that can be serviced by a (κ,μ)-wise independent hash function that maps Δ_h into R and uses h probes or less. Let T be a lower bound for the number of probes that are necessary, in the worst case, for a (κ,μ)-wise independent function that services D. Then a lower bound for the total number of probes necessary to service all of D is $|D|T-|\Delta_{T-1}|-|\Delta_{T-2}|-\cdots-|\Delta_1|$, since the negative terms comprise overcorrections for the numbers of probes that are miscounted by the principal term $|D|T$. The

result now follows from Theorem 3.1—which says that $\frac{1}{2}\Delta_d < \frac{z^{\frac{d}{}}}{(\kappa-2)^{\frac{d-1}{}}}$, provided $\mu \leq |R|/2$—and from division by $|D|$. □

Of course Theorem 3.1 also gives an admissible estimate to use for T.

Theorem 3.1 suggests that T has a very mild dependence on the parameter μ, and our constructions have typically allowed μ to remain at 1. On the other hand, it is worth noticing that the lower bound exhibits a far greater dependence on $|D|$, which could be dramatically larger than $|R|$. In such cases, we can use a simple, randomly chosen linear congruence (cf. F_0^k in the appendix) to premap $|D|$ into, say, R^4, whence the resource requirements dictated by Theorem 3.1 would depend on $|R|^4$ rather than $|D|$. Thus there is a significant benefit, in some cases, in allowing an asymptotically negligible fraction of hash functions to be defective, due to distinct data values colliding at the premapping stage. Corollary 3.2 proves that unless this possibility of failure is permitted, the storage requirement for the family will be of size $|D|^\delta$ for some suitable $\delta < 1$, instead of $|R|^\delta$.

Finally, it is worth observing that Theorem 3.1 can be used quite simply to give a moderately satisfactory lower bound for the running time of r-practical hash functions, where this prehashing step is used to reduce the effective size of the domain.

COROLLARY 3.3. *Suppose $|D| \gg |R|^{r/2}$, and let F be a family of hash functions F with domain D and range R that is r-practical (κ, μ)-wise independent. Then*

$$z^{\frac{T}{}} > (\kappa-2)^{\frac{T-1}{}} \frac{|R|^{r/2}}{2}\left(1 - \frac{\mu}{|R|}\right).$$

Proof. The inequality follows as a straightforward application of Theorem 3.1. □

Although the exponent $r/2$ in this bound is surely too small by a factor of 2, its influence on the probe count T is still fairly acceptable. Notice that if the data can be examined during the function selection stage, then the prehashing step can be made one to one, and Theorem 3.1 will be applicable with D replaced by the reduced domain. The resulting function family, in this case, can then be $(\kappa, 1)$-wise independent without appealing to r-practicality.

4. A space-time tradeoff and problem equivalence. We have already observed the dramatic change in the requirements for z when d, the number of random seeds read per hash function evaluation, drops from κ to $\kappa - 1$. However, the lower bound in Theorem 3.1 requires some effort to decode. Suppose that $\mu < |R|/2$ and $\kappa > 3$. We can weaken (3.9) to the statement $z^d > |D|$. Let $|D| = n^k$ and $z = n^\epsilon$. It follows that $d > \frac{k}{\epsilon}$ is a necessary condition for d.

Theorem 2.10 gives a matching sufficiency result. With the same domain and array size parameters as above, the formulation reads

$$\epsilon \geq \frac{2k}{d} + \frac{1 + \ln d + \ln \kappa}{\ln n}.$$

We can require that $\ln n \gg \ln \kappa$ and view the last term as insignificant. In this case, the number of probes to the auxiliary array of z random words that is necessary per evaluation of a (κ)-wise independent hash function that maps $[0, n^k - 1]$ into $[0, n^\ell - 1]$ satisfies

$$d = \Theta(k/\epsilon) \quad \text{for } d < \kappa.$$

Restated, we have a time-log(space) tradeoff: $d \times \log(CacheSize) \geq \log(DomainSize)$, where $CacheSize$ is the dimension z of the random word array A, which contains

words from R, and $DomainSize = |D|$. This lower bound and tradeoff apply to any algorithm including those with random precomputation (that is oblivious to the data) as long as any internal storage and precomputed values are counted as part of the array as measured by z. Furthermore, the program need not be uniform, and the storage and time used by a deterministic precomputation that precedes the reading of the random seeds need not be charged in this space-time tradeoff. (The point of these remarks is that the tradeoff need not include the generation of G, which could be probabilistic or based on exhaustive search, which would entail an extravagant use of time and space, but result in an optimally structured graph.)

From a more abstract perspective, we have exposed a very close equivalence between the operation count T_f for evaluating (κ)-wise independent hash functions that map, say, $[0, n-1]$ into $[0, n-1]$, and the operation count T_G needed to compute the neighbors of an input vertex of bipartite graphs on $[0, n-1] \times [0, n^\epsilon - 1]$ that have low outdegree d and good expansion properties for small vertex sets. A spatially compact graph representation that can be used to compute the adjacency list of an input vertex in time $T_G = cd$ gives a time $T_f \approx cT_G$ hash function with a high degree of independence, when augmented with an array of n^ϵ random numbers. Similarly, a set of ϵd functions that are independent and are selected from a (κ)-wise independent family can be used to construct such an adjacency list in time $T_G \approx \epsilon d T_f$, albeit with an additive spatial cost of $\epsilon d n^\epsilon$ for the random number arrays: The equivalence holds in this direction because our probability estimates in section 2 were calculated from κ-way expectations, and never used full independence. The resource blowup is the modest factor ϵd because a random function value in $[0, n-1]$ gives $\frac{1}{\epsilon}$ points in $[0, n^\epsilon - 1]$ (where all expressions, for expositional simplicity, are assumed to be integers). A crude application of our lower bound imposes the requirement that $d > 1/\epsilon$, while our existential Theorem 2.10 establishes sufficiency for $d = 2/\epsilon + 1$, provided $\log \kappa \ll \log n$.

Taken together, these upper and lower bounds show that up to a fairly fine granularity of instruction counting, the computational complexities of these two algorithms problems are within a factor of 2 of each other.

5. Formal applications. Because of the graphs needed to construct explicit $O(\log n)$-wise independent constant-time hash functions, all applications must be presently viewed as theoretical. Nevertheless, the constructions in section 2 show that (κ)-wise independent hash functions, for nonconstant $\kappa < n^\delta$ and sufficiently small constant $\delta > 0$, can, in principle, be programmed as constant-time subroutines. Thus, we have established the formal feasibility of any probabilistic algorithm that is based on these functions.

The most fundamental application for hash functions is, of course, to compute randomized mappings of data. Specific applications where constant-time computations and high independence are needed can be found in [9, 11, 16]. Additional uses can be found at http://citeseer.nj.nec.com/context/122560/0. In the next three subsections, we give additional examples.

The hashing applications of section 5.1 are immediate, although the referenced papers are necessary to see that the hash functions are adequate for the strong bounds we report.

Section 5.2 establishes a variety of results about distributed hashing with universal classes that exhibit $O(\log n)$-wise independence. Section 5.3 presents one of the main consequences for parallel processing, which is that constant-time $O(\log n)$-wise independent hash functions enable Ranade's PRAM emulation scheme to be run

on an $n \times \log n$ butterfly network with just one column of processors and memory modules. Each processor has $\log n$ pipelined processes, and the machine comprising n processors, n memory modules, and $n \log n$ switches can emulate a T-time, $n \log n$ processor common PRAM algorithm with an expected running time of $O(T \log n)$. Thus, the processor efficiency is optimal, up to constant factors, although $(T \log^2 n)$ switching operations are required. No other results are known for emulating the common PRAM with an optimal speedup (in processor cycles) and a comparable degree of parallel processing.

However, for brevity, we omit any discussion about how to restructure Ranade's algorithm to work in this new setting; the specifics are straightforward, and can be deduced from the original algorithm [20]. Instead, we focus on the generic randomization arguments for the performance analysis. In some cases, similar results have appeared elsewhere (cf. [35]), although the method of proof has sometimes precluded the use of our hash functions.[5] For purposes of achieving full generality, all of the proofs in this section will be based solely on the independence characteristics of universal classes of hash functions.

5.1. Hashing. Our first application concerns double hashing, uniform hashing, and linear probing. The performances of these hashing schemes with $O(\log n)$-wise independent functions are analyzed in an elaborate proof that begins in [22] and is completed in [23, 25]. In view of these results, the following is immediate.

COROLLARY 5.1. *For any fixed load factor $\alpha < 1$, $O(\log n)$-wise independent hash functions can be used for double hashing with constant time per probe and an expected cost of $\frac{1}{1-\alpha} + O(\frac{1}{n})$ probes for unsuccessful search.*

The results extend directly to uniform hashing, and the optimality of such functions for linear probing is likewise direct, given the developments in [25]. All previous analyses required the data to be random, or used idealized, fully random hash functions, which cannot be effectively programmed.

5.2. The statistics of distributed hashing. The bulk of this section concerns the statistical analysis of distributed hashing with separate chaining in a model of limited independence.

DEFINITION 5.2. *Let the model M denote the following scenario. The hash function f is randomly selected from a family of $(\kappa, 1)$-wise independent hash functions with domain and range $D = [0, nm - 1]$. The data set S comprises all mn elements in D, and is hashed by f. The range $[0, mn - 1]$ is distributed across n modules, each with an address space of size m. Keys that hash to the same location are linked together via separate chaining within the module.*

There are no limitations on the size of m, although we implicitly presume that the hashing and comparison of address locations require a constant number of operations.

The point of hashing all of D is to establish conservative performance bounds based on collision chains that have sustained full growth. The underlying randomization assumes that the hash function is chosen prior to the program execution and, barring failure, is used throughout the computation. A failure occurs if f maps too many domain elements to a given memory module. Such events require a new hash function to be chosen and the parallel program to be run afresh.

[5] In particular, some proofs exploit the fact that a degree k hashing polynomial has at most k roots in a finite field. The requisite randomness can require that $k = \Theta(\log n)$. For our hashing formulations, the number of keys hashing to a single value can be much larger, and can even be the entire data set.

All data references are presumed to be independent of the hash function.

For specificity, let module j store the keys hashing into $[jm, (j+1)m-1]$. In separate chaining, a key x that is the first to hash to a location $f(x)$ is stored there. Subsequent keys hashing to $f(x)$ are called colliders, and are inserted into the chain headed by x. The standard algorithm stores all colliders in memory that is external to the hash-based address space. There are many variants; it is also possible to store colliders in vacant locations within the table, which, absent relocation procedures, will accelerate the chain growth (in what is known as coalesced hashing). Alternatively, it is possible to relocate colliders when new items are hashed to their current locations. Typically, the initialization overhead is ignored, since it is possible to initialize a hash table in constant time via software techniques (cf. [1, Prob. 2.12, p. 71]) or in hardware. We will follow the usual conventions of ignoring initialization overhead and storing colliders outside of the hash space.

The total storage requirement for a module, therefore, is the number of keys that hash into its address space plus the number of vacant locations that remain in the address space once all of $[0, mn-1]$ is hashed. The statistics for these two random variables are established in Lemmas 5.3 and 5.4. The first result is standard and is included for completeness. The second seems to have received less attention for models of universal hashing.

The performance statistics for data access will be based on processing $n \log n$ memory references with modules servicing their hash-based requests in parallel independently of the other modules. The reason we analyze $n \log n$ requests is that $cn \log n$ data accesses, for $c > 1$, will turn out to have a performance that can be modeled as c sets of $n \log n$ requests. For significantly smaller collections, the variance in behavior for each module will degrade the overall performance. Restated, the size $s = n \log n$ is, up to constant factors, the smallest value that will have an expected parallel service time of $\Theta(\frac{s}{n})$, which is optimal.

An analysis of storage requirements begins with the observation that a fully random hashing of mn elements will distribute, on average, m elements to each module. Typically, $m \gg n$, but the uniformity requirements will just need m to be at least $g \log n$ for some large constant g. The result will be that with high probability, $\Theta(\log n)$-wise independence is adequate to distribute the mn keys fairly evenly among all n modules.

LEMMA 5.3. *In model M, let d_i be the size of the preimage of $[im, (i+1)m-1]$, so that $d_i = |\{x : f(x) \in [im, (i+1)m-1]\}|$. Let $\kappa = \beta \log n$, where β is a positive constant, and suppose that $m \geq \kappa$. Then for any fixed $c > 0$, there is a fixed d such that $\text{Prob}\{d_i > dm\} \leq \frac{1}{n^c}$. Furthermore, for any fixed $d > 1$ and fixed c, there is a suitably large constant β such that for any $m > \frac{\kappa}{d-1}$, $\text{Prob}\{d_i > dm\} \leq \frac{1}{n^c}$ in model M.*

Proof. Suppose $d \geq 1 + \epsilon + \frac{\kappa}{m}$. The $(\kappa, 1)$-wise independence facilitates the straightforward calculation

$$\begin{aligned}
\text{Prob}\{d_i > dm\} &= \text{Prob}\left\{\binom{d_i}{\kappa} > \binom{dm}{\kappa}\right\} \\
&\leq \frac{\text{E}[\binom{d_i}{\kappa}]}{\binom{(1+\epsilon)m+\kappa}{\kappa}} \\
&\leq \frac{\binom{nm}{\kappa}\frac{1}{n^\kappa}}{\binom{(1+\epsilon)m+\kappa}{\kappa}} < \frac{(nm)^\kappa \frac{1}{n^\kappa}}{((1+\epsilon)m)^\kappa} = \left(\frac{1}{1+\epsilon}\right)^\kappa.
\end{aligned}$$

Setting $(1+\epsilon) = 2^{c/\beta}$ establishes the first bound.

The second bound follows from setting $\epsilon = (d-1) - \frac{\kappa}{m}$, observing that $\frac{\kappa}{m} < (d-1)$ by supposition, and choosing β so large that $(1+\epsilon) = 2^{c/\beta}$. □

The next task is to bound, with high probability, the number of keys that will be hashed into the auxiliary storage of a given module. Since Lemma 5.3 ensures that with high probability, no more than $(1+\epsilon)m$ keys are distributed to each module, the remaining task is to determine how many locations in a module are left vacant. As before, our expectations are based on the easy analysis for models with full randomness.

Suppose that n items are assigned integer values in the range $[0, n-1]$, with the assignments uniformly distributed and equally likely. The probability that a fixed integer is not assigned to any item is $(1-\frac{1}{n})^n \approx \frac{1}{e}$. It follows that the expected number of vacant locations in a module ought to be about $\frac{m}{e}$, if all mn values are hashed. The next lemma shows that with $(\kappa \log n)$-wise independent hash functions, this statistic is about right, and bounds the probability of large deviations.

LEMMA 5.4. *In model M, let $\alpha > 0$ and $\epsilon > 0$ be fixed constants, and suppose that $m > e\alpha^2 \log^2 n$. Let V be the number of address locations in the hash space of module 1 that remain vacant after all of $[0, mn-1]$ has been hashed. Suppose that $\kappa \geq (1 + e^{1+1/e})\alpha \log n$. Then*

$$\text{Prob}\{V > (1+\epsilon)\frac{m}{e} < 4\left(\frac{1}{1+\epsilon}\right)^{\alpha \log n}.$$

Proof. The proof has two steps. We show that $(e^{1+1/e}\alpha \log n)$-wise independence is adequate to ensure that any fixed set of $\alpha \log n$ locations in the hash table will be vacant with a probability below $2e^{-\alpha \log n}$. Given the aforementioned probability, we bound $\text{Prob}\{V \geq (1+\epsilon)\frac{m}{e}\}$ by estimating the number of $(\alpha \log n)$-tuples of locations in module 1 that are vacant. This latter bound is established as follows:

$$\begin{aligned}
\text{Prob}\left\{\binom{V}{\alpha \log n} \geq \binom{(1+\epsilon)\frac{m}{e}}{\alpha \log n}\right\} &\leq \frac{\binom{m}{\alpha \log n} \times 2e^{-\alpha \log n}}{\binom{(1+\epsilon)\frac{m}{e}}{\alpha \log n}} \\
&\leq \frac{m^{\alpha \log n} 2e^{-\alpha \log n}}{((1+\epsilon)m/e)^{\alpha \log n} \prod_{j=0}^{\alpha \log n - 1}\left(1 - \frac{je}{(1+\epsilon)m}\right)} \\
&\leq \frac{1}{1 - \frac{e\alpha^2 \log^2 n}{2m}}\left(\frac{e}{1+\epsilon}\right)^{\alpha \log n} 2e^{-\alpha \log n} \\
&\leq 4\left(\frac{1}{1+\epsilon}\right)^{\alpha \log n}.
\end{aligned}$$

It is clear that this formulation requires $\alpha \log n$ degrees of freedom in the form of hashing independence.

As for the first calculation, let X be the event that a given set of $\alpha \log n$ locations are vacant. Let $t = \lceil e^{1+1/e}\alpha \log n \rceil$.

From basic inclusion-exclusion (and limited independence) it follows that

$$\text{Prob}\{X\} = \sum_{0 \leq j < t} (-1)^j \binom{mn}{j}\left(\frac{\alpha \log n}{nm}\right)^j + \delta(-1)^t \binom{mn}{t}\left(\frac{\alpha \log n}{nm}\right)^t$$

for some δ satisfying $0 < \delta < 1$. Consequently,

$$\left|\text{Prob}\{X\} - \left(1 - \frac{\alpha \log n}{mn}\right)^{mn}\right| < \binom{mn}{t}\left(\frac{\alpha \log n}{nm}\right)^t < \left(\frac{e\alpha \log n}{t}\right)^t < e^{-\alpha \log n}$$

(and a stronger Stirling estimate for $t!$ would show that the difference is actually less than $\frac{e^{-\alpha \log n}}{\sqrt{t}}$). Applying the triangle inequality to the previous display gives

$$\text{Prob}\{X\} < \left(1 - \frac{\alpha \log n}{mn}\right)^{mn} + e^{-\alpha \log n}.$$

Since $(1 - \frac{|a|}{x})^x$ is increasing in x, it follows that $(1 - \frac{\alpha \log n}{mn})^{mn} < e^{-\alpha \log n}$, and step 1 is now complete. Altogether, the calculations used $t + \alpha \log n$ degrees of freedom. □

Lemmas 5.3 and 5.4 show that for suitable $\kappa = O(\log n)$, each module can have a storage capacity of $(1 + \epsilon)(1 + 1/e)m$ records, and the probability of a hash-induced overflow will be polynomially small, for large enough $\kappa = O(\log n)$. The hashing model is just traditional separate chaining, and no auxiliary hash functions are needed. Of course, the excess storage can be reduced to ϵm by other storage procedures.

The remaining issue is to bound the processing time for an arbitrary collection of $n \log n$ data references. Lemma 5.5 begins this process by proving that an arbitrary set of $n \log n$ references will be distributed among the n modules fairly evenly.

LEMMA 5.5. *In model M, let $\widehat{D}$ be a set of $n \log n$ address keys in D. Let $\widehat{d}_i$, for $i = 0, 1, \ldots, n-1$, be the number of keys in $\widehat{D}$ that are hashed into $[im, (1+i)m - 1]$, so that $\widehat{d}_i = |\{x \in \widehat{D} : im \le f(x) < (i+1)m\}|$. Suppose $\kappa \ge \log n$. Then for any $d > 3$,*

$$\text{Prob}\{\widehat{d}_i > d \log n\} \le \frac{1}{(d-1)^{\log n}}.$$

Proof.

$$\text{Prob}\{\widehat{d}_i \ge d \log n\} \le \frac{\binom{n \log n}{\log n}\left(\frac{1}{n}\right)^{\log n}}{\binom{d \log n}{\log n}} \le \frac{(\log n)^{\log n}}{(d \log n)^{\underline{\log n}}} < \frac{1}{(d-1)^{\log n}}. \qquad \square$$

By itself, Lemma 5.5 says little about the cost of memory accesses when separate chaining is used to store the data. We need to bound the expected sum of the accessed chain lists to determine the access time. The bound will be established in two steps. Lemma 5.6 shows that the sum of $\log n$ chain lengths is, with high probability and on average, just $O(\log n)$. The proof will not assume a priori limitations on the number of items that might collide to a given location; the count could even be mn. Lemma 5.7 extends Lemma 5.6 to include the random number of chains that must be traversed in each module. The chief issue is to establish this accommodation without materially affecting the size requirements for κ.

LEMMA 5.6. *In model M, let κ be at least $2 \log n$. Let $\widehat{D_j} = \{z_i\}_{i=1}^{\log n}$ be a set of $\log n$ memory references that hash to module j. Let Δ_i comprise the elements of $D - z_i$ that collide with z_i under f : $\Delta_i = \{x \in D : f(x) = f(z_i) \bigwedge x \ne z_i\}$. Let $d^* = \sum_{i \le \log n}(|\Delta_i|)$. Let c be a fixed constant greater than 3. Then*

$$\text{E}[d^* \cdot \mathcal{X}(d^* > c \log n)] = O\left(\frac{\log n}{(c-1)^{\log n}}\right).$$

Proof. Counting the number of $(\log n)$-tuples in $\cup_{i=1}^{\log n}\Delta_i$ shows that

$$\begin{aligned}\mathrm{E}[d^*\cdot\mathcal{X}(d^*>c\log n)] &\le (\log n)\mathrm{E}\left[\frac{\binom{d^*}{\log n}}{\binom{c\log n}{-1+\log n}}\right]\\ &\le \frac{(\log n)\binom{mn}{\log n}\left(\frac{\log n}{mn}\right)^{\log n}}{\binom{c\log n}{-1+\log n}}\\ &\le \frac{(\log n)^{\log n}}{((c-1)\log n)^{-1+\log n}}\\ &\le \frac{\log n}{(c-1)^{-1+\log n}}. \qquad \square\end{aligned}$$

The reason that κ is taken to be at least $2\log n$ is to ensure that the requirement that $\widehat{D_j}$ hashes to one module does not impart any conditioning to the hashing statistics for samples of $\log n$ keys in $D-\widehat{D_j}$.

Markedly simple facts about random variables will suffice to extend the bound of Lemma 5.6 to larger numbers of chain lengths. In particular, let $X_1, X_2, \ldots, X_\ell$ be a sequence of arbitrary (possibly dependent) random variables.

Recall that $\mathrm{E}[\sum_i X_i] = \sum_i \mathrm{E}[X_i]$.

Observe that for $a \ge 0$, $\mathrm{E}[\max_i(X_i\cdot\mathcal{X}(X_i\ge a))] \le \sum_i \mathrm{E}[X_i\cdot\mathcal{X}(X_i\ge a)]$.

Finally, let $Y=\sum_{1\le i\le\ell}X_i$, and suppose that the X_i are identically distributed. Then for $a\ge 0$, $Y\cdot\mathcal{X}(Y\ge\ell a)\le\sum_i(X_i\cdot\mathcal{X}(X_i\ge a)+a\sum_{j\ne i}\mathcal{X}(X_j>a))$ because the right-hand side is an overestimate of what would result from rounding X_i up to the value a in the event that $X_i<a$ and $X_j>a$ for some $j\ne i$. Hence

$$\begin{aligned}\mathrm{E}[Y\cdot\mathcal{X}(Y\ge\ell a)] &\le \ell\mathrm{E}[X_1\cdot\mathcal{X}(X_1\ge a)]+\ell(\ell-1)a\,\mathrm{Prob}\{X_1>a\}\\ &\le \ell^2 E[X_1\cdot\mathcal{X}(X_1\ge a)].\end{aligned} \tag{5.1}$$

As unsatisfactory as these bounds may seem, they are tight in the worst case. (However, there are more appealing formulations when, for example, the expectations are replaced by standard overestimates.)

These simple facts, plus Lemmas 5.5 and 5.6, are sufficient to bound the fraction of time spent executing exceptionally slow parallel data accesses.

LEMMA 5.7. *In model M, let $\widehat{D}$ be a set of $n\log n$ address keys in D. Let $\widehat{\Delta}_i$ be the elements in $\widehat{D}$ that hash to memory module i : $\widehat{\Delta}_i=\{x\in\widehat{D} : im\le f(x)<(i+1)m\}$. Let $\widehat{d}_i=|\widehat{\Delta}_i|$. Let Δ_i be the elements in $D-\widehat{D}$ that hash to locations in $\{f(\widehat{\Delta}_i)\}$, and set $\delta_i=|\Delta_i|$. Let $\delta=\max_i\delta_i$. Suppose that $\kappa\ge 2\log n$.*

Then for any integer $c\ge 2$,

$$\mathrm{E}[\delta\cdot\mathcal{X}(\delta\ge(4c+1)(8c+1)\log n)] = O\left(\frac{1}{c^{\log n}}\right).$$

Proof. According to Lemma 5.5,

$$\mathrm{Prob}\{\widehat{d}_i\ge(8c+1)\log n\}<\frac{1}{n^3c^{\log n}} \quad \text{for } i=1,2,\ldots,n.$$

If $\widehat{d}_i\le(8c+1)\log n$, then δ_i is statistically dominated by $Y_1+Y_2+\cdots+Y_\ell$, where $\ell=8c+1$, and each Y_i has the same distribution as d^* in Lemma 5.6. Because of

limited independence, we cannot assume that the Y_i are even pairwise independent. However, it follows from inequality (5.1) and Lemma 5.6 that as long as $\widehat{d}_i$ is bounded by $\ell \log n$,

$$\mathrm{E}[\delta_i \cdot \mathcal{X}(\delta_i > \ell(4c+1)\log n)] \leq \frac{4c\ell^2 \log n}{n^2 c^{\log n}}.$$

So if $\widehat{d}_i \leq \ell \log n$ for $i = 0, \ldots, n-1$, then

$$\mathrm{E}[\max_i(\delta_i \cdot \mathcal{X}(\delta_i > \ell(4c+1)\log n))] \leq n\frac{4c\ell^2 \log n}{n^2 c^{\log n}} \leq \frac{4c\ell^2}{c^{\log n}}.$$

If some $\widehat{d}_i$ is too large, we bound the resulting time by $\sum_i \delta_i$, which is as if all of $\widehat{D}$ were hashed to just one module. This sum is statistically dominated by $Y = Y_1 + Y_2 + \cdots + Y_n$. Since the probability that some $\widehat{d}_i$ is too large is bounded by $n\frac{1}{n^3 c^{\log n}}$, no estimate is needed for the event that some $\widehat{d}_i$ is too large and $Y \leq n^2$. We need only observe that $\mathrm{E}[Y \cdot \mathcal{X}(Y > n^2)] \leq n^2 \mathrm{E}[d^* \cdot \mathcal{X}(d^* > n)]$, and this latter expectation is insignificant by Lemma 5.6.

Last, we note that these calculations impose no requirements on κ, apart from the restrictions resulting from applications of Lemmas 5.5 and 5.6. In fact, the requisite independence is just the maximum of the requirements for κ as used in these two lemmas. To verify this requirement for κ, we need to bound $\mathrm{E}[\max_i(\delta_i \cdot \mathcal{X}(\delta_i > \ell(4c+1)\log n))]$ by a sum of expectations that are each bounded by $O(\frac{1}{c^{\log n}})$, provided $\kappa \geq 2\log n$. We do so as follows:

$$\begin{aligned}
&\max_i(\delta_i \cdot \mathcal{X}(\delta_i > \ell(4c+1)\log n)) \\
&\quad \leq \mathcal{X}(\forall i : \widehat{d}_i \leq \ell \log n) \cdot \max_i(\delta_i \cdot \mathcal{X}(\delta_i > \ell(4c+1)\log n)) \\
&\qquad + \mathcal{X}(\exists i : \widehat{d}_i > \ell \log n) \cdot n^2 \cdot \mathcal{X}\left(\sum_{i \leq n} \delta_i \leq n^2\right) + \left(\sum_{i \leq n} \delta_i\right) \cdot \mathcal{X}\left(\sum_i \delta_i > n^2\right),
\end{aligned}$$

whence

$$\begin{aligned}
&\mathrm{E}[\max_i(\delta_i \cdot \mathcal{X}(\delta_i > \ell(4c+1)\log n))] \\
&\quad \leq n\ell^2 \mathrm{E}[d^* \cdot \mathcal{X}(d^* \geq (4c+1)\log n)] \\
&\qquad + n^3 \operatorname{Prob}\{\widehat{d}_1 > \ell \log n\} + n^2 \mathrm{E}[d^* \cdot \mathcal{X}(d^* > n)],
\end{aligned}$$

where unnecessary constraints have been dropped to reduce the independence requirements for the last three terms on the right. □

Together, these lemmas show that in model M, $O(\log n)$-wise independent hash functions can be used to implement distributed hashing with separate chaining with high efficiency.

THEOREM 5.8. *In model M, let $\kappa = \beta \log n$ for some constant $\beta \geq 2$. Then a universal class of $(\kappa, 1)$-wise independent hash functions can be used to implement distributed hashing with separate chaining so that $n \log n$ data access requests will have, on average and with overwhelming probability (i.e., probability $1 - 1/n^c$, for any constant c), a parallel service time of $\Theta(\log n)$, which is optimal for systems with n independent memory modules. Furthermore, suppose that $m \gg \log^2 n$. Then given $\epsilon > 0$, there is a fixed $\beta > 0$ such that a family of $(\beta \log n, 1)$-wise independent*

hash functions will, on average and with overwhelming probability, distribute the data among the modules with a density that is, up to a factor of $1+\epsilon$*, no worse than the expected distribution that results from fully random hashing.*

Proof. The first statement follows from Lemmas 5.5 and 5.7. In particular, Lemma 5.5 shows that each module receives only $\Theta(\log n)$ of the $n \log n$ hash keys on average, and with overwhelming probability. Lemma 5.7 shows that the sum of the hash chain lengths for $\Theta(\log n)$ addresses is $\Theta(\log n)$ on average and with overwhelming probability. Lemma 5.7 also affirms that in the overwhelmingly improbable case where some module receives more than $\Theta(\log n)$ of the $n \log n$ hash keys, the sum of the chain lengths will, on average and with overwhelming probability, be bounded by $O(n^2)$, which is statistically insignificant since by Lemma 5.5, the probability that at least $(d+1)\log n$ of the $n \log n$ hash keys are mapped to one module, for $d > 2$, is bounded by $\frac{n}{n^{\log d}}$.

The second assertion summarizes Lemmas 5.3 and 5.4.

The inclusion-exclusion argument of Lemma 5.4 is sensitive to small statistical errors, which requires that μ be set to 1. Since the hashing application uses the same domain and range, no prehashing function (as described in section 2) is necessary, and μ can be set to 1. □

It is worth noting that the requirement that $\mu = 1$ is unnecessary for the first part of Theorem 5.8. Setting $\mu > 1$ would cause the derived performance results to degrade by just a factor of μ. However, the bounds for the memory requirements would not hold for $\mu > 1$, although they would only change by a small constant factor. As presented, Lemma 5.7 requires an independence as high as $\kappa = \beta \log n$ with $\beta = 2$. However, the decomposition method used to establish Lemma 5.7 can be applied to give corresponding results for any constant $\beta > 0$, and the resulting performance results would degrade by constant factors.

5.2.1. Hashing failures. For models of randomized computation, and for distributed randomized computation in particular, some of the complications associated with hashing failures are worthy of analysis.

Our model of computation assumes that a hash-based program can be rerun afresh if the hashing for some reason turns out to be unsuccessful. There are several different policies that can be used to manage the consequences of hashing failures. The random variables X and C will represent the execution time of a randomized program. In particular, X will be the running time of a probabilistic program that completes its execution successfully with a hash function that is randomly selected from a family of κ-wise independent functions. C can model the additional execution time that is used when the program completes successfully but has some kind of nonfatal irregularity in the construction of hash function family. Alternatively, C can represent the additional time used in an unsuccessful execution of the program when there is no preemptive termination of the processing.

The following random variables extend these formulations to running times for different processing policies, including some where programs are terminated prematurely. Such programs are then run afresh with new random seeds. Technically, the following would be defined in terms of random stopping times. However, program formulations give equivalent definitions with more transparent semantics.

Define $R(X;C)$ via the following program.
Let X_i and C_i be independent identically distributed copies of X and C.
$sum \leftarrow 0$;
$i \leftarrow 0$;

Repeat
  $i \leftarrow i + 1$;
  $sum \leftarrow sum + X_i + C_i$;
Until $C_i = 0$;
$R(X; C) \leftarrow sum$.

Define $F(X; C; \rho)$ via the following program.
Let X_i and C_i be independent identically distributed copies of X and C.
$sum \leftarrow 0$;
$i \leftarrow 0$;
Repeat
  $i \leftarrow i + 1$;
  $sum \leftarrow sum + \min(X_i + C_i, \rho)$;
Until $X_i + C_i \leq \rho$;
$F(X; C; \rho) \leftarrow sum$.

Define $G(X; C; \rho)$ via the following program.
Let X_i and C_i be independent identically distributed copies of X and C.
$i \leftarrow 1$;
$sum \leftarrow 0$;
Repeat
  $i =\leftarrow i + 1$;
  $sum \leftarrow sum + \min(\rho^i, X_i + C_i)$;
Until $X_i + C_i \leq \rho^i$;
$G(X; C) \leftarrow sum$.

The program execution policies corresponding to these random variables are as follows. R models running a probabilistic program to termination repeatedly until the execution is successful. F models running the program repeatedly for ρ time steps and then starting afresh until an execution is successful. G corresponds to running the ith trial of a program for ρ^i time steps and repeating the effort afresh with trial $i + 1$ until a trial executes successfully.

Section 2 presented several types of hash functions that exhibit different types of performance degradation depending on what goes wrong. If a prehashing function h is used to reduce the effective size of the domain D, then a (mild) failure would occur if two keys in the data have the same image under h. Of course prehashing failures could be much worse.

If the hash function uses a probabilistically constructed expander-like graph, then a failure can occur if the graph does not achieve the required characteristics. As mentioned in section 5.2, a failure can also occur in a distributed model of computation if, for any reason, the hashing does not distribute the data evenly enough to satisfy the physical storage limitations for each module. This event is a fatal distributional failure. Another type of distributional irregularity would occur if the chain lengths (that result from hashing with separate chaining) have a distribution that deviates significantly from the predictions of Lemma 5.7. Similarly, a distributional irregularity can occur in a distributed model of computation if the module access patterns (as characterized by Lemma 5.5) are highly deviant.

These sources of performance inefficiency can be modeled as follows.

Let d be a positive constant. For $k = 1, 2, \ldots, T$, let P_k be a random variable satisfying $\mathrm{E}[P_k] = \tau$ and $\mathrm{E}[\max(0, P_k - d\tau)] < \frac{\tau}{n^r}$. Each P_j will represent the processing time for a single block of $n \log n$ data access requests in model M in the case that the underlying family of hash functions satisfies the requirements of the model. These random variables can be dependent.

To model failures that erode parallel processing, let A be a random variable

satisfying $\text{Prob}\{A = 0\} = 1 - \frac{1}{n^{r+1}}$ and $\text{Prob}\{A = nT\tau\} = \frac{1}{n^{r+1}}$. In the rare event that $A \neq 0$, A will turn out to be n times slower than the expected time for a T-step parallel program, and will correspond to the case where each instruction is executed sequentially with no concurrent processing.

To model more catastrophic failures, let $B(\epsilon)$ for $0 < \epsilon < 1$ be a random variable satisfying $\text{Prob}\{B = 0\} = \epsilon$ and $\text{Prob}\{B = \infty\} = 1 - \epsilon$. $B(1 - \frac{1}{n^r})$ gives a very conservative overestimate of performance losses due to severe deficiencies in the construction of the hash function.

Given these definitions, the performance consequences of hashing irregularities are now readily quantified.

LEMMA 5.9. *Let $(y)^+ = \max(y, 0)$. Let the execution styles R, F, and G be as defined above. Let A, B, $P_1, P_2, \ldots, P_T$ be random variables satisfying the following:*

$\text{Prob}\{A = 0\} = 1 - \frac{1}{n^{r+1}}$, $\text{Prob}\{A = nT\tau\} = \frac{1}{n^{r+1}}$;

$\text{Prob}\{B = 0\} = \epsilon$, $\text{Prob}\{B = \infty\} = 1 - \epsilon$;

$\text{E}[P_k] = \tau$ *and* $\text{E}[\max(0, P_k - d\tau)] < \frac{\tau}{n^r}$.

Let $X = \sum_{k=1}^{T} P_k$.

Then

(i) $\text{E}[R(X; A)] = \Theta(T\tau)$.

$\text{E}[(R(X; A) - dT\tau)^+] = O(\frac{T\tau}{n^r})$.

$\text{E}[F(X; B(1 - \frac{1}{n^r}); (d+1)T\tau)] = \Theta(T\tau)$.

For any constant $s \geq 1$,

$\text{E}[(F(X; B(1 - \frac{1}{n^r}); (d+1)T\tau) - s(d+1)T\tau)^+] = O(\frac{T\tau}{n^{rs}})$.

$\text{E}[G(X; B(1 - \frac{1}{n^r}); 2)] = O(T\tau)$.

$\text{E}[(G(X; B(1 - \frac{1}{n^r}); 2) - 4dT\tau)^+] = O(\frac{T\tau}{n^r})$.

(ii) *For any constant* ϵ, *where* $0 < \epsilon < 1$,

$\text{E}[F(X; B(\epsilon); (d+1)T\tau)] = \Theta(T\tau)$.

For any constant ρ, *there is a constant* σ *such that*

$\text{E}[(F(X; B(\epsilon); (d+1)T\tau) - \sigma T\tau \log n)^+] = O(\frac{T\tau}{n^\rho})$.

For any constant ρ, *where* $1 < \rho < \frac{1}{1-\epsilon}$, $\text{E}[G(X; B(\epsilon); \rho)] = \Theta(T\tau)$.

Proof. (i) By definition of X,

$$\text{E}[X] = \sum_{k=1}^{T} \text{E}[P_k] = T\tau.$$

The first assertion in (i) is now immediate since

$$\text{E}[R(X, A)] = T\tau + (T\tau + nT\tau) \sum_{i=1}^{\infty} \left(\frac{1}{n^{r+1}}\right)^i,$$

where $T\tau$ corresponds to the expected time for the successful execution, and $T\tau + nT\tau$ bounds the expected time used in each unsuccessful execution.

Since $\max(0, x)$ is convex,

$$\text{E}[\max(0, X - dT\tau)] \leq \sum_{i=1}^{T} \text{E}[\max(0, P_k - d\tau)]$$

by Jensen's inequality. So

$$\text{E}[\max(0, X - dT\tau)] \leq \sum_{i=1}^{T} \frac{\tau}{n^r} = \frac{T\tau}{n^r}.$$

By definition, $R(X, A)$ corresponds to i consecutive samples of X and A with $A \neq 0$ in each sample, followed by a sample where $A = 0$, for a probabilistically determined i. So

$$\mathrm{E}[\max(0, R(X; A) - dT\tau)] \leq \mathrm{E}[\max(0, X - dT\tau)] + \sum_{i=1}^{\infty} (\mathrm{E}[X] + nT\tau) \left(\frac{1}{n^{r+1}}\right)^i$$
$$= O\left(\frac{T\tau}{n^r}\right).$$

The extension of these results to $F(X; B(1 - \frac{1}{n^r}); (d+1)T\tau)$ is fairly straightforward. Note that $\mathrm{Prob}\{X+B > (d+1)T\tau\} \leq \mathrm{Prob}\{B \neq 0\}+\mathrm{Prob}\{X > (d+1)T\tau\}$, and that $\mathrm{Prob}\{X > (d+1)T\tau\} \leq \frac{\mathrm{E}[\max(0, X - dT\tau)]}{T\tau} \leq \frac{1}{n^r}$. So $\mathrm{Prob}\{X + B > (d+1)T\tau\} \leq \frac{2}{n^r}$. Consequently,

$$\mathrm{E}\left[\max\left(0, F\left(X; B\left(1 - \frac{1}{n^r}\right); (d+1)T\tau\right) - s(d+1)T\tau\right)\right] \leq \sum_{i=s}^{\infty} ((d+1)T\tau)\left(\frac{2}{n^r}\right)^i$$
$$= O\left(\frac{T\tau}{n^{rs}}\right).$$

As for policy G, the sum of the geometric progression is comparable to its largest term, and once the time trials are as large as $(d+1)T\tau$, the probability of a successful completion is $1 - O(\frac{1}{n^r})$ per iteration.

(ii) The proofs are straightforward and are omitted. □

Policy G was designed to facilitate computations where the expected execution time of a program is unknown. The error B was defined to accommodate the possibility that a hash function might be highly defective but still partition the keys so evenly that the storage limitation for each module is not exceeded. The application of Lemma 5.9 to hashing is straightforward, and is included in the performance analysis of section 5.3.

5.3. A pipelined version of Ranade's PRAM algorithm. The results of section 5.2 help establish that constant-time hash functions can be used in a pipelined version of Ranade's PRAM emulation architecture.

Another stepping-stone in proving optimal speedup concerns the fast sorting of random numbers and is independent of the underlying hashing, apart from the fact that κ-wise independence results in κ distinct elements being assigned statistically random values.

The need for linear-time sorting is due to the step in Ranade's algorithm where each row in the network uses its $\log n$ processors for an $O(\log n)$-time systolic bubblesort of the row's $\log n$ locally generated hash values. If each row is to have one processor available for sorting the same data, an optimal scheme will (on average) have just $\Theta(\log n)$ time available for the slowest row to complete this sorting step.

As stated below, Lemma 5.11 (with $\log n$ substituted for n) will ensure that for any $\alpha > 0$, there is a d such that each of the n pipelined processors can (after straightforward preprocessing has combined identical hash values) sort its $\log n$ local hash packets in $d \log n$ steps with a probability that exceeds $1 - \frac{1}{n^\alpha}$.

Let S be a set of n integers in the range $[0, m-1]$. For $s \in S$, let $s_{top} = \lfloor n\frac{s}{m} \rfloor$ be the $\log_2 n$ most significant bits of s. Suppose that s_{top} can be computed in constant time and that any two elements $s, t \in S$ can be compared in constant time. We make no other assumptions about the relationship between m and n.

DEFINITION 5.10. *Let the following sorting procedure be* Algorithm S.

For each $s \in S$, insert s into Bin$[s_{top}]$.

For $j \leftarrow 0$ to $n-1$, Mergesort the contents of Bin$[j]$.

Concatenate the contents of the sorted Bins.

The mean performance of Algorithm S is, as the next lemma states, within a constant factor of optimal.

LEMMA 5.11. *Let S be a random sequence of n integers selected uniformly from $[0, m-1]$. Then for any fixed $c > 0$, there is a fixed d such that Algorithm S sorts a fraction exceeding $1-e^{-cn}$ of all $S \in [0, m-1]^n$ in dn steps, where $c = d-2-\ln(d-1)$.*

Proof. See [27, Theorem 4.1, p. 208]. □

All of the data processing characteristics are now established. We now define a kind of best-case performance characteristic for networks and then combine all of the results in the optimal performance formulation of Corollary 5.13.

DEFINITION 5.12. *Let $\mathcal{N}$ be a network. We say that a* no-load memory reference *takes time t on $\mathcal{N}$ if, in the worst case, a single memory access (with all other process requests suspended) can be executed on $\mathcal{N}$ in a worst-case time of t.*

COROLLARY 5.13. *Let Ω be a pipelined n-processor analogue of Ranade's architecture that includes an $n \times \log n$ butterfly switching network. Let each processor execute a $\log n$-deep pipeline that corresponds to a row of processors in Ranade's organization, and manage a memory module of $O(m)$ words. Suppose that a no-load memory reference can be processed on Ω in $O(\log n)$ time.*

Then a T-time $n \log n$-processor common PRAM algorithm for mn words of shared memory can be emulated on Ω in an expected time of $O(T \log n)$. Furthermore, for any fixed r, there is a fixed d such that the algorithm runs on Ω in a time that is bounded by $dT \log n$ with probability exceeding $1 - \frac{1}{n^r}$. The hash function can be selected from a family of uniformly r-practical $(\beta \log n, 1)$-wise independent hash functions for fixed $\beta \geq 8$, and separate chaining can be used to resolve collisions as in model M. In addition, if $m \gg \log^2 n$, and, for fixed $\epsilon > 0$, β is set to a sufficiently large constant, then the requisite auxiliary storage will, with overwhelming probability, be within a factor of $1+\epsilon$ of the expected storage used by fully random hash functions.

Proof. The performance of the algorithm is a consequence of its probability of failure and the running time of each step of the algorithm in the event that no failure occurs.

We first consider the performance when the hash function meets the requirements of model M.

The PRAM program is executed in blocks of $n \log n$ instructions and has execution times for each of the T instruction blocks. Let R_j be the round-trip network routing time for the jth block of instructions as defined (and analyzed) in Ranade's emulation algorithm [20]. The analysis requires that β be at least 8. Let S_j, for the jth block of instructions, be the time used for the concurrent local sortings of the n sets of $\log n$ hash-based packet addresses as required in Ranade's routing algorithm, implemented by Algorithm S and analyzed in Lemma 5.11. Let M_j be the time expended executing the concurrent memory references for the n modules as required by the jth block of instructions. We take the fixed overhead associated with each $(n \log n)$-instruction block to require $O(\log n)$ time and presume that it is absorbed by the other random variables.

Let

$$X = \sum_{j=1}^{T} (R_j + S_j + M_j),$$

so that X accounts for the running time of the pipelined PRAM emulation, provided the hashing meets the requirements of Ranade's theorem and model M. Ranade's analysis, Lemma 5.11, and Theorem 5.8 ensure that $\mathrm{E}[X] = \Theta(T \log n)$. Similarly, these analyses show that for any r there is a d such that for all j, $\mathrm{E}[S_j \cdot \mathcal{X}(S_j > d \log n)] < \frac{1}{n^r}$, and likewise for all R_j and M_j.

Even if the hashing uses $(\kappa, 1)$-wise independent hash functions, failure can occur if too many keys hash to some module. In this case, the algorithm must be started afresh with a new hash function. We might anticipate that if an emulation attempt does not run to completion, then it ought to have statistics that are dominated by those of X. However, it is possible that a failed computation could comprise the execution of almost all T blocks of instructions, and that each round might be somewhat slower than that for a successful execution because some module is almost out of memory. Thus, a failure can skew the running time to be worse than average. We need only argue that the expected performance, given that such a failure occurs, does not degrade by more than a factor of n, which corresponds to the worst-case event where all data hashes to just one module and the amount of data (up to the last instruction block) just fills up all of the available storage. We can assume that the expected length (and sum of $n \log n$ lengths) of the collision chains is still governed by the randomness of the hash functions, since these values are not changed by more than a small constant factor when the amount of data in a module is at its maximum. The probability that such a hashing scheme fails to distribute the data adequately well among the modules is, by Lemma 5.3, polynomially small. Thus, the emulation scheme in this case (of ∞-practical hashing) has a performance that is modeled by $R(X, A)$ in Lemma 5.9 of section 5.2.

If the underlying hash function depends on a randomized graph construction of section 2, then the graph could be defective with a probability of $\frac{1}{(mn)^r}$. Formally, this form of hashing failure lies within the model of uniformly r-practical $(\kappa, 1)$-wise independent hash functions as quantified by Definition 2.1.

In the worst case, the physical storage restrictions for each module would be satisfied, but the collision chains would be unacceptably long. In such a circumstance, we must presume that $\Theta(m)$ keys hash to a single location in each module. Even these delays cause an expected performance degradation that is nominal. Alternatively, the delays can be viewed as infinite, modeled by the error random variable $B(1 - \frac{1}{n^r})$ and accommodated by emulation policies F or G. Lemma 5.9 shows that in all cases, the resulting performance has the statistical characteristics asserted in this corollary.

Last, the claims about the storage requirements are just a restatement of facts established in section 5.2. □

As in Theorem 5.8, none of these performance characteristics, apart from the tight bounds on the requisite storage for each module, require μ to be 1. For larger values of μ all of the other bounds would again degrade by just a factor of μ, and the requisite auxiliary storage would increase by a small constant factor.

Of course, any reasonable strategy can be used to resolve the case of module overflow since its probability of occurrence is statistically insignificant. Similarly, the data in each memory module can be stored via any reasonable secondary hashing scheme.

Remarks. Although the main result of this paper suggests that there is little additional cost in creating constant-time hash functions that are far more random than $O(\log n)$-wise independent, our analyses of distributed hashing and PRAM emulation have followed a more parsimonious path by restricting the randomness allocated to

these functions to be $\beta \log n$ for a fixed value of β. The reasons for this restriction are twofold. First, the sufficiency of $O(\log n)$-wise independence gives hope that even less randomness might suffice for real implementations. Second, the realization of fast random functions really awaits the discovery of suitable graphs; it is conceivable (and indeed likely) that good graphs might first be found for more moderate degrees of randomness, such as those which give $O(\log n)$-wise independence. Of course, the results of this section are just feasibility proofs, and not a blueprint for future architectures even if ideal graphs are found.

For completeness, we note that Kruskal, Rudolph, and Snir use pipelines of depth n^ϵ to get a parallel emulation time that is optimal up to a factor of $\frac{1}{\epsilon}$ [13]. Versions of some of the basic counting estimates can also be found elsewhere [13, 18, 20, 35]. For example, formulations comparable to Lemmas 5.5 and 5.6 can be found in [20]. Ranade uses the fact that polynomials of degree $\log n$ have at most $\log n$ roots to bound the preimage size to $O(\log n)^2$ for the fetches that must be executed by a single row of $\log n$ processors. Valiant uses the root bound, parallel sorting, and secondary hashing (with separate chaining) to control the fetch time for a variety of architectures [35].

It should also be noted that the fast Fourier transform can be used to compute k evaluations of a degree k polynomial in $O(k \log^2 k)$ steps (cf. [1, Cor. 2, p. 224]). Thus, it is possible to use the above pipeline strategy on n processors with $\log n$ degree hash functions to attain a performance cost of $O(\log \log n)^2$ operations per memory reference rather than a naive $\log n$. We have shown that this multiplicative performance penalty can, in theory, be reduced to an asymptotic $O(1)$.

6. Conclusions. The high independence exhibited by our hash functions enriches the class of probabilistic algorithms that can be shown to achieve their expected performance in formal models of real computation. Proofs need not be restricted to (κ)-wise independence for constant κ, and probability estimates can use the probabilistic method to calculate the expected number of κ-tuples satisfying various behavior criteria for these larger values of κ.

On the other hand, it is worth noting that the fast hash functions described in this paper are not really necessary for pure routing problems. After all, if an adequately random assignment of intermediate destinations provides, with very high probability, nearly optimal performance in a Valiant–Brebner style of routing [33], then the same destinations could be used for many consecutive routings.

What these fast hash functions really provide are nearly uniform random mappings of data and a convenient way to assert that with high probability, various maldistributions are very unlikely to occur. In addition, hash functions computed from destination addresses provide a way for common memory references to be fully combined en route in Ranade's elegant queue management scheme, and this might be important if combining is required to avoid hot spot contention.

But while the asymptotic characteristics of fast hash functions are now well understood, the feasibility question remains wide open. Without doubt, the most significant open problem is to find good local expander-like graphs that are defined by short efficient programs. The discovery of such structures might have a very beneficial effect on the practicality of these classes of functions. Interestingly, recent progress suggests that this prospect might be within reach [4, 14, 21].

Appendix. For completeness, we state the two combinatorial lemmas that are needed to analyze the basic prehashing functions of section 1.1.

Fact 1 (cf. [17, 8]). Let $P_k = \{p \mid p \text{ is prime and } p \in (n^k \ln m, (2+\beta)n^k \ln m)\}$ for some small suitably fixed $\beta > 0$. Then

$$\forall x \neq y \in [0, m-1] : \text{Prob}_{p \in P_k}\{x = y \bmod p\} < n^{-k}.$$

Proof [17, 8]. The prime number theorem says that $|P_k| = \frac{(1+\beta)n^k \ln m}{k \ln n + \ln \ln m}(1 - o(1))$, whence fewer than $1/n^k$ of the elements of P_k can divide $|x - y|$, since $\prod_{p \in P_k} p > (n^k \ln m)^{|P_k|} > (m)^{n^k}$. □

Fact 2 (cf. [5]). Let $F_0(p) = \{h \mid h(x) = (ax + b \bmod p) \bmod n^k,\ a \neq 0, b \in [0, p-1]\}$, where $p > n^k$ is prime. Then

$$\forall x \neq y \in [0, p-1] : \text{Prob}_{f \in F_0(p)}\{f(x) = f(y)\} \leq n^{-k}.$$

Proof [5]. Given x and y, $x, y \in [0, p-1]$, $x \neq y$, the number of different $f \in F_0(p)$, where $f(x) = f(y)$, is precisely the number of 2×2 linear systems in a and b:

$$\begin{cases} ax + b = c + dn^k \bmod p \\ ay + b = c + en^k \bmod p \end{cases} \qquad c, d, e \geq 0;\ c + dn^k < p;\ c < n^k;\ e \neq d;\ c + en^k < p.$$

This system is designed so that $c + dn^k$ can have p different values: c ranges from 0 to $n^k - 1$, and dn^k gives increments of n^k. The same is true of $c + en^k$, since it has the same format. Both expressions equal c when taken modn^k, and all possible values are captured by this representation. The parameters e and d cannot be equal because the solution to the system would then give $a = 0$. Furthermore, a cannot be zero if $e \neq d$, which is required for x and y to be distinct. Straightforward counting shows that there are at most $\lceil p/n^k - 1 \rceil$ different values available for e. Since there are $p(p-1)$ different functions in F_0, and $f(x) = f(y)$ for at most $p\lceil p/n^k - 1 \rceil \leq p\frac{p-1}{n^k}$ of them, the result follows. □

Combining Facts 1 and 2 shows that a hash function selected at random from $F_0^k = \cup_{p \in P_k} F_0(p)$ will, with probability exceeding $1 - 2\binom{n}{2}n^{-k}$, map n items from $[0, m-1]$ into $[0, n^k - 1]$ with no collisions at all among its $\binom{n}{2}$ pairs. We could take $k = 4$, so that the probability of a collision is below $1/n^2$, and assume the functions $F_0(p)$ are defined as in Fact 2 for $p \approx n^4 \ln m$.

Acknowledgments. It is a pleasure to thank the two anonymous referees for their many helpful suggestions, which have improved the presentation and clarity of this paper. Similarly, I would like to thank the editorial chain of SICOMP for their effort on behalf of this submission.

REFERENCES

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] R. Aleliunas, *Randomized parallel communication*, in Proceedings of the 13th ACM-SIGOPS Symposium on Principles of Distributed Computing, 1982, pp. 60–72.

[3] C.H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, *Experimental quantum cryptography*, J. Cryptology, 5 (1992), pp. 3–28.

[4] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson, *Randomness conductors and constant-degree expansion beyond the degree/2 barrier*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 659–668.

[5] J.L. Carter and M.N. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[6] A. Chin, *Locality-preserving hash functions for general purpose parallel computation*, Algorithmica, 12 (1994), pp. 170–181.
[7] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R.E. Tarjan, *Dynamic perfect hashing: Upper and lower bounds*, SIAM J. Comput., 23 (1994), pp. 738–761.
[8] M.L. Fredman, J. Komlós, and E. Szemerédi, *Storing a sparse table with $O(1)$ worst case access time*, J. ACM, 31 (1984), pp. 538–544.
[9] L.A. Goldberg, Y. Matias, and S. Rao, *An optical simulation of shared memory*, SIAM J. Comput., 28 (1999), pp. 1829–1847.
[10] A. Karlin and E. Upfal, *Parallel hashing: An efficient implementation of shared memory*, J. ACM, 35 (1988), pp. 876–892.
[11] R. Karp, M. Luby, and F. Meyer auf der Heide, *Efficient PRAM simulation on a distributed memory machine*, Algorithmica, 16 (1996), pp. 517–542.
[12] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed., Addison-Wesley, Reading, MA, 1998.
[13] C.P. Kruskal, L. Rudolph, and M. Snir, *A complexity theory of efficient parallel algorithms*, Theoret. Comput. Sci., 71 (1990), pp. 95–132.
[14] C.-J. Lu, O. Reingold, S. Vadhanm, and A. Wigderson, *Extractors: Optimal up to constant factors*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003, pp. 602–611.
[15] G.S. Lueker and M. Molodowitch, *More analysis of double hashing*, Combinatorica, 13 (1993), pp. 83–96.
[16] Y. Matias and A. Schuster, *Fast, efficient mutual and self simulations for shared memory and reconfigurable mesh*, Parallel Algorithms Appl., Special Issue on Algorithms for Enhanced Mesh Architectures, 8 (1996), pp. 195–221.
[17] K. Mehlhorn, *Data Structures and Efficient Algorithms* 1: *Sorting and Searching*, Springer-Verlag, Berlin, Heidelberg, 1984.
[18] K. Mehlhorn and U. Vishkin, *Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories*, Acta Inform., 21 (1984), pp. 339–374.
[19] N. Pippenger, *Parallel communication with limited buffers*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, 1984, pp. 127–136.
[20] A.G. Ranade, *How to emulate shared memory*, J. Comput. System Sci., 42 (1991), pp. 307–326.
[21] O. Reingold, S. Vadhan, and A. Wigderson, *Entropy waves, the zig-zag graph product, and new constant-degree expanders*, Ann. of Math. (2), 155 (2002), pp. 157–187.
[22] J.P. Schmidt and A. Siegel, *The analysis of closed hashing under limited randomness*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, 1990, pp. 224–234.
[23] J.P. Schmidt and A. Siegel, *Double Hashing Is Computable and Randomizable with Universal Hash Functions*, SIAM J. Comput., submitted.
[24] J.P. Schmidt, A. Siegel, and A. Srinivasan, *Chernoff–Hoeffding bounds for applications with limited independence*, SIAM J. Discr. Math., 8 (1995), pp. 223–250.
[25] A. Siegel and J.P. Schmidt, *Closed Hashing Is Computable and Optimally Randomizable with Universal Hash Functions*, SIAM J. Comput., submitted.
[26] A. Siegel, *On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 20–25.
[27] A. Siegel, *Median bounds and their application*, J. Algorithms, 38 (2001), pp. 184–236.
[28] B. Smith, *A pipelined, shared resource MIMD computer*, in Proceedings of the International Conference on Parallel Processing, International Association for Computers and Communications, 1978, pp. 6–8.
[29] J. Spencer, *Ten Lectures on the Probabilistic Method*, 2nd ed., CBMS-NSF Regional Conf. Ser. in Appl. Math. 64, SIAM, Philadelphia, 1994.
[30] E. Upfal, *Efficient schemes for parallel computation*, J. ACM, 31 (1984), pp. 507–517.
[31] E. Upfal, *A probabilistic relation between desirable and feasible models of parallel computation*, in Proceedings of the 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 258–265.
[32] E. Upfal and A. Wigderson, *How to share memory in a distributed system*, J. ACM, 34 (1987), pp. 116–127.
[33] L.G. Valiant and G.J. Brebner, *Universal schemes for parallel communication*, in Proceedings of the 13th Annual ACM Symposium on Theory of Computing, 1981, pp. 263–277.
[34] L.G. Valiant, *Optimally universal parallel computers*, Phil. Trans. Royal Soc. London Ser. A, 326 (1988), pp. 373–376.

[35] L.G. Valiant, *General purpose parallel architectures*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 943–972.
[36] M.N. Wegman and J.L. Carter, *New hash functions and their use in authentication and set equality*, J. Comput. System Sci., 22 (1981), pp. 265–279.
[37] A.C. Yao, *Uniform hashing is optimal*, J. ACM, 32 (1985), pp. 687–693.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 544–562

LOCAL SEARCH HEURISTICS FOR k-MEDIAN AND FACILITY LOCATION PROBLEMS*

VIJAY ARYA†, NAVEEN GARG†, ROHIT KHANDEKAR†, ADAM MEYERSON‡, KAMESH MUNAGALA‡, AND VINAYAKA PANDIT§

Abstract. We analyze local search heuristics for the metric k-median and facility location problems. We define the *locality gap* of a local search procedure for a minimization problem as the maximum ratio of a locally optimum solution (obtained using this procedure) to the global optimum. For k-median, we show that local search with swaps has a locality gap of 5. Furthermore, if we permit up to p facilities to be swapped simultaneously, then the locality gap is $3 + 2/p$. This is the first analysis of a local search for k-median that provides a bounded performance guarantee with only k medians. This also improves the previous known 4 approximation for this problem. For uncapacitated facility location, we show that local search, which permits adding, dropping, and swapping a facility, has a locality gap of 3. This improves the bound of 5 given by M. Korupolu, C. Plaxton, and R. Rajaraman [*Analysis of a Local Search Heuristic for Facility Location Problems*, Technical Report 98-30, DIMACS, 1998]. We also consider a capacitated facility location problem where each facility has a capacity and we are allowed to open multiple copies of a facility. For this problem we introduce a new local search operation which opens one or more copies of a facility and drops zero or more facilities. We prove that this local search has a locality gap between 3 and 4.

Key words. local search, approximation algorithm, facility location

AMS subject classifications. 68W25, 90B80, 90C59

DOI. 10.1137/S0097539702416402

1. Introduction.

The problem of locating facilities in a manner so that they can effectively serve a set of clients has been the subject of much research. While one could consider fairly general measures of effectiveness of a set of locations in serving the clients, one measure that is typically used is the distance between the client and the facility that is serving it. Since by opening a lot of facilities, we can be near every client, it also makes sense to take into account the number of facilities opened in judging the quality of a solution. These two measures, typically referred to as the service cost and the facility cost, can be combined in many ways to obtain interesting variants to the general facility location problem. For instance, in *k-median* we require that at most k facilities be opened and the total service cost, measured as the sum of the distance of each client to the nearest open facility, be minimum. Instead of setting a limit on the total number of facilities that could be opened, we sometimes associate with every facility a cost of opening that facility. The facility cost of a solution is then the sum of the costs of the facilities that are opened, and the quality of the solution is measured by the sum of the facility and service costs. This, in fact, is the

*Received by the editors October 21, 2002; accepted for publication (in revised form) October 30, 2003; published electronically March 23, 2004. A preliminary version of the paper appeared in *Proceedings of the 33rd ACM Symposium on Theory of Computing*, Crete, Greece, 2001.
http://www.siam.org/journals/sicomp/33-3/41640.html

†Department of Computer Science and Engineering, Indian Institute of Technology, New Delhi 110016, India (vijayarya@cse.iitd.ernet.in, naveen@cse.iitd.ernet.in, rohitk@cse.iitd.ernet.in). The first author was partially supported by an IBM Fellowship. The third author was partially supported by an Infosys Fellowship.

‡Department of Computer Science, Stanford University, Stanford, CA 94305 (awm@cs.ucla.edu, kamesh@cs.stanford.edu). The fourth author was supported by ARO DAAG-55-97-1-0221. The fifth author was supported by ONRN00014-98-1-0589.

§IBM India Research Lab, Block I, Indian Institute of Technology, New Delhi 110016, India (pvinayak@in.ibm.com).

classical *facility location problem*. Note that in this setting the facility costs need not be the same and would, in general, depend on the location at which the facility is being opened. A generalization of the classical facility location problem arises when we associate a capacity with each facility, which measures the maximum number of clients that the facility can serve. Further variants of this *capacitated facility location* (CFL) problem arise when we bound the number of facilities that can be opened at a certain location. Thus in k-CFL, we can open at most k facilities at any location.

Local search techniques have been very popular as heuristics for hard combinatorial optimization problems. The 1-exchange heuristic by Lin and Kernighan [15] for the metric-TSP remains the method of choice for practitioners. However, most of these heuristics have poor worst-case guarantees, and very few approximation algorithms that rely on local search are known. Könemann and Ravi [13] used local search algorithms for degree-bounded minimum spanning trees. Chandra, Karloff, and Tovey [3] show an approximation factor of $4\sqrt{n}$ for the 2-exchange local search heuristic for the Euclidean traveling salesman problem. Khuller, Bhatia, and Pless [12] give a local search approximation algorithm for finding a feedback edge-set incident upon the minimum number of vertices. Local search has also been used for set packing problems by Arkin and Hassin [2]. Here, we provide worst-case analysis of local search algorithms for facility location problems.

For an instance I of a minimization problem, let $\texttt{global}(I)$ denote the cost of the global optimum and $\texttt{local}(I)$ be the cost of a locally optimum solution provided by a certain local search heuristic. We call the supremum of the ratio $\texttt{local}(I)/\texttt{global}(I)$ the *locality gap* of this local search procedure. For 1-CFL with uniform capacities, Korupolu, Plaxton, and Rajaraman [14] argued that any procedure that permits adding, dropping, or swapping a facility has a locality gap of at most 8. Their analysis was subsequently refined and tightened by Chudak and Williamson [8] to yield a locality gap of at most 6. Pál, Tardos, and Wexler [19] present a local search algorithm for 1-CFL with nonuniform capacities which has a locality gap of 9. Mahdian and Pál [16] considered the universal facility location problem where the cost of opening a facility is any arbitrary but monotone function of the demand that the facility serves; note that this problem generalizes k-CFL. Mahdian and Pál extended the algorithm of [19] to obtain a local search algorithm with a locality gap of 8. For uncapacitated facility location (UFL), Korupolu, Plaxton, and Rajaraman [14] provided a bound of 5 on the locality gap when the only operations permitted are those of adding, dropping, or swapping a facility. Charikar and Guha [4] introduced an operation which permits adding a facility and dropping many and showed that this local search procedure has a locality gap of exactly 3. For k-median, Korupolu, Plaxton, and Rajaraman [14] gave a local search procedure which permitted adding, deleting, and swapping facilities and gave a solution with $k(1+\epsilon)$ facilities having a service cost at most $3+5/\epsilon$ times the optimum k-median solution.

A different approach to facility location was employed by Shmoys, Tardos, and Aardal [20] and Charikar et al. [5]. They formulated the problems as linear programs and rounded the optimum fractional solution to obtain a 3 approximation for the UFL problem and a $6\frac{2}{3}$ approximation for k-median. Jain and Vazirani [11] gave an alternate 3 approximation algorithm for UFL using the primal-dual schema. They also observed that UFL can be viewed as a Lagrange-relaxation of k-median and utilized this to give a 6 approximation algorithm for k-median. Later, Charikar and Guha [4] improved this to a 4 approximation. Recently, Archer, Rajagopalan, and Shmoys [1] showed that the algorithm due to Jain and Vazirani can be made to satisfy the "continuity" property and established an integrality gap of at most 3 for the most

natural LP relaxation for the k-median problem. However, their proof gives only an exponential time algorithm. Guha and Khuller [9] employed randomization to improve the approximation guarantee of UFL to 2.408. This was further improved to $(1+2/e)$ by Chudak [6] and finally to 1.728 by Charikar and Guha [4]. Similar ideas were used by Chudak and Shmoys [7] to obtain a 3 approximation algorithm for ∞-CFL when the capacities are uniform. Jain et al. [10] used the method of dual fitting and factor revealing LP to design two greedy algorithms for the UFL problem with approximation guarantees of 1.861, 1.61 and running times of $O(m \log m)$, $O(n^3)$, respectively, where n is the number of vertices and m is the number of edges in the underlying graph. Mahdian, Ye, and Zhang [17] combined the ideas in [10] with the idea of cost scaling to obtain an approximation factor of 1.52 for UFL, which is also the best known. Jain and Vazirani [11] obtained a 4 approximation algorithm for ∞-CFL when the capacities were nonuniform by solving a related UFL problem using their primal-dual algorithm. Recently, Mahdian, Ye, and Zhang [18] gave a 2 approximation for the ∞-CFL with nonuniform capacities by reducing it to a linear-cost facility location problem.

Our results. In this paper, we analyze local search heuristics for three problems.

1. For metric k-median, we show that the local search with single swaps has a locality gap of 5. This is the first analysis of a local search for k-median that provides a bounded performance guarantee with only k medians. We also show that doing multiple swaps, that is, dropping at most p facilities and opening the same number of new facilities, yields a locality gap of $3 + 2/p$. This improves on the 4 approximation algorithm for k-median by Charikar and Guha [4]. Our analysis of the locality gap is tight; that is, for an infinite family of instances there is a locally optimum solution whose service cost is nearly $(3 + 2/p)$ times that of the global optimum.
2. For metric UFL, we show that local search, which permits adding, dropping, and swapping a facility, has a locality gap of 3. This improves the bound of 5 given by Korupolu, Plaxton, and Rajaraman [14]. Again, our analysis of the algorithm is tight. We show how our algorithm can be improved to achieve a $1 + \sqrt{2} + \epsilon \approx 2.414 + \epsilon$ approximation using ideas from [4].
3. For metric ∞-CFL, we consider the setting when the capacities may be nonuniform and argue that local search, where the only operation permitted is to add multiple copies of a facility and drop zero or more facilities, has a locality gap of at most 4. We give a polynomial time algorithm that uses `Knapsack` as a subroutine to search for a lower cost solution in the neighborhood. We also show that there is a locally optimum solution with cost 3 times the optimum. We show how our algorithm can be improved to achieve a $1 + \sqrt{3} + \epsilon \approx 3.732 + \epsilon$ approximation using ideas from [4].

The paper is organized as follows. Section 2 introduces some notation and defines the problems addressed in this paper formally. In section 3, we first prove a locality gap of 5 for the k-median problem when only single swaps are permitted and then extend the analysis to argue a locality gap of $3 + 2/p$ when up to p facilities can be swapped simultaneously. Sections 4 and 5 discuss the algorithms for UFL and ∞-CFL, respectively. Section 6 concludes with some open problems.

2. Notation and preliminaries. In the k-median and facility location problems, we are given two sets: F, the set of *facilities*, and C, the set of *clients*. Let $c_{ij} \geq 0$ denote the cost of serving client $i \in C$ by a facility $j \in F$; we will think of this as the distance between client i and facility j. The goal in these problems is to

identify a subset of facilities $S \subseteq F$ and to serve all clients by facilities in S such that some objective function is minimized. The facilities in S are said to be *open*. The metric versions of these problems assume that distances c_{ij} are symmetric and satisfy the triangle inequality. The problems considered in this paper are defined as follows.

1. The metric k-median problem. Given integer k, identify a set $S \subseteq F$ of at most k facilities to open such that the total cost of serving all clients by open facilities is minimized.
2. The metric UFL problem. For each facility $i \in F$, we are given a cost $f_i \geq 0$ of opening the facility i. The goal is to identify a set of facilities $S \subseteq F$ such that the total cost of opening the facilities in S and serving all the clients by open facilities is minimized.
3. The metric ∞-CFL problem. For each facility $i \in F$, we are given a cost $f_i \geq 0$ of opening a copy of facility i and an integer capacity $u_i > 0$, which is the maximum number of clients that a single copy of the facility i can serve. The output is a set of facilities $S \subseteq F$ and the number of copies of each facility in S to be opened. The goal is to serve each client by a copy of a facility in S such that the number of clients served by a copy of a facility is at most its capacity. The objective is to minimize the total facility cost and the cost of serving all the clients.

Thus for all the problems we consider, a solution can be specified by giving the set of open facilities together with their multiplicities. In the rest of this paper we will think of a solution as a multiset of facilities.

Algorithm `Local Search.`

1. $S \leftarrow$ an arbitrary feasible solution in $\mathcal{S}$.
2. While $\exists\, S' \in \mathcal{B}(S)$ such that $cost(S') < cost(S)$,
   do $S \leftarrow S'$.
3. return S.

FIG. 1. *A generic local search algorithm.*

A generic local search algorithm (Figure 1) can be described by a set $\mathcal{S}$ of all feasible solutions, a cost function $cost : \mathcal{S} \to \mathbb{R}$, a *neighborhood* structure $\mathcal{B} : \mathcal{S} \to 2^{\mathcal{S}}$, and an oracle that, given any solution S, finds (if possible) a solution $S' \in \mathcal{B}(S)$ such that $cost(S') < cost(S)$. A solution $S \in \mathcal{S}$ is called *locally optimum* if $cost(S) \leq cost(S')$ for all $S' \in \mathcal{B}(S)$; the algorithm in Figure 1 always returns one such solution. The cost function and the neighborhood structure $\mathcal{B}$ will be different for different problems and algorithms.

If S is a locally optimum solution, then for all $S' \in \mathcal{B}(S)$,

$$cost(S') - cost(S) \geq 0.$$

Our proof of the locality gap proceeds by considering a suitable, polynomially large (in the input size) subset $\mathcal{Q} \subseteq \mathcal{B}(S)$ of neighboring solutions and arguing that

$$\sum_{S' \in \mathcal{Q}} (cost(S') - cost(S)) \leq \alpha \cdot cost(O) - cost(S),$$

where O is an optimum solution and $\alpha > 1$ a suitable constant. This implies that $cost(S) \leq \alpha \cdot cost(O)$, which gives a bound of α on the locality gap.

To translate the proof of the locality gap into an approximation algorithm with polynomial running time, we modify step 2 of the algorithm as follows.

2. While $\exists\, S' \in \mathcal{B}(S)$ such that $cost(S') \leq (1 - \epsilon/Q)\ cost(S)$, do $S \leftarrow S'$.

Here $\epsilon > 0$ is a constant and $Q = |\mathcal{Q}|$. Thus, in each local step, the cost of the current solution decreases by a factor of at least ϵ/Q. If O denotes an optimum solution and S_0 denotes the initial solution, then the number of steps in the algorithm is at most $\log(cost(S_0)/cost(O))/\log\frac{1}{1-\epsilon/Q}$. As $Q, \log(cost(S_0))$, and $\log(cost(O))$ are polynomial in the input size, the algorithm terminates after polynomially many local search steps.

To analyze the quality of this locally optimum solution S, we note that for all $S' \in \mathcal{Q}$, $cost(S') > (1 - \epsilon/Q)cost(S)$. Hence

$$\alpha \cdot cost(O) - cost(S) \geq \sum_{S' \in \mathcal{Q}} (cost(S') - cost(S)) > -\epsilon \cdot cost(S),$$

which implies that $cost(S) \leq \frac{\alpha}{(1-\epsilon)} cost(O)$. Thus our proof that a certain local search procedure has a locality gap of at most α translates into an $\alpha/(1-\epsilon)$ approximation algorithm with a running time that is polynomial in the input size and $1/\epsilon$.

We use the following notation. Given a solution A, let A_j denote the *service cost* of client j, which is the distance between j and the facility in A which serves it. For every facility $a \in A$, we use $N_A(a)$ to denote the set of clients that a serves (Figure 2). For a subset of facilities, $T \subseteq A$, let $N_A(T) = \bigcup_{a \in T} N_A(a)$.

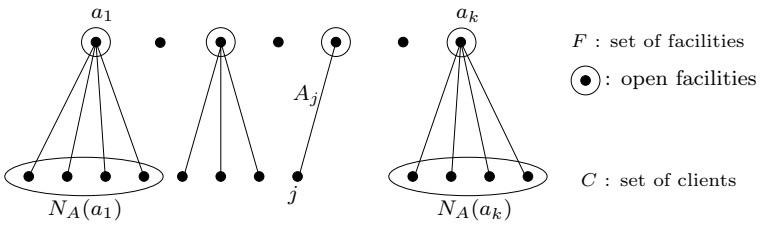


FIG. 2. *Illustration of neighborhood and service costs.*

3. The *k*-median problem. The k-median problem is to open a subset $S \subseteq F$ of at most k facilities so that the total service cost is minimized. Thus, if a client j is served by a facility $\sigma(j) \in S$, then we wish to minimize $cost(S) = \sum_{j \in C} c_{j\sigma(j)}$. For a fixed S, serving each client by the nearest facility in S minimizes this cost.

3.1. Local search with single swaps. In this section, we consider a local search using single swaps. A swap is effected by closing a facility $s \in S$ and opening a facility $s' \notin S$ and is denoted by $\langle s, s' \rangle$; hence $\mathcal{B}(S) = \{S - \{s\} + \{s'\} \mid s \in S\}$. We start with an arbitrary set of k facilities and keep improving our solution with such swaps until we reach a locally optimum solution. The algorithm is described in Figure 1. We use $S - s + s'$ to denote $S - \{s\} + \{s'\}$.

3.2. The analysis. We now show that this local search procedure has a locality gap of 5. Let S be the solution returned by the local search procedure and let O be an optimum solution. From the local optimality of S, we know that

$$cost(S - s + o) \geq cost(S) \quad \text{for all } s \in S, o \in O. \tag{1}$$

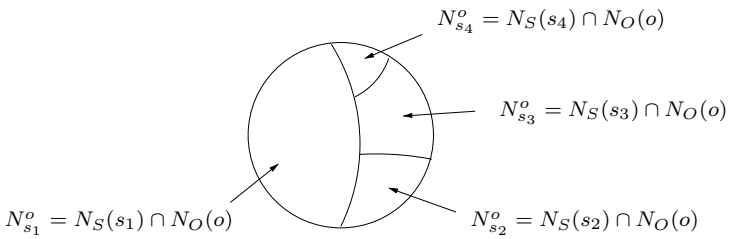


FIG. 3. *Partitioning* $N_O(o)$.

Note that even if $S \cap O \neq \emptyset$, the above inequalities hold. We combine these inequalities to show that $cost(S) \leq 5 \cdot cost(O)$.

Consider a facility $o \in O$. We partition $N_O(o)$ into subsets $N^o_s = N_O(o) \cap N_S(s)$ as shown in Figure 3.

DEFINITION 3.1. *We say that a facility* $s \in S$ captures *a facility* $o \in O$ *if* s *serves more than half the clients served by* o*, that is,* $|N^o_s| > \frac{1}{2}|N_O(o)|$.

It is easy to see that a facility $o \in O$ is captured by at most one facility in S. We call a facility $s \in S$ *bad*, if it captures some facility $o \in O$, and *good* otherwise. Fix a facility $o \in O$ and consider a 1-1 and onto function $\pi : N_O(o) \to N_O(o)$ satisfying the following property (Figure 4).

PROPERTY 3.1. *If* s *does not capture* o*, that is,* $|N^o_s| \leq \frac{1}{2}|N_O(o)|$*, then* $\pi(N^o_s) \cap N^o_s = \emptyset$.

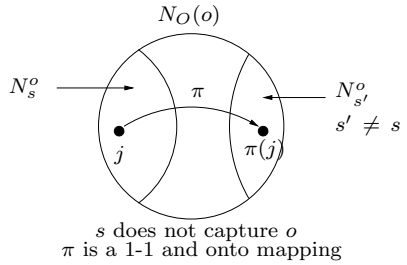


FIG. 4. *The mapping* π *on* $N_O(o)$.

We outline how to obtain one such mapping π. Let $D = |N_O(o)|$. Order the clients in $N_O(o)$ as $c_0, \ldots, c_{D-1}$ such that for every $s \in S$ with a nonempty N^o_s, the clients in N^o_s are consecutive; that is, there exists p, q, $0 \leq p \leq q \leq D-1$, such that $N^o_s = \{c_p, \ldots, c_q\}$. Now, define $\pi(c_i) = c_j$, where $j = (i + \lfloor D/2 \rfloor)$ modulo D. For contradiction assume that both $c_i, \pi(c_i) = c_j \in N^o_s$ for some s, where $|N^o_s| \leq D/2$. If $j = i + \lfloor D/2 \rfloor$, then $|N^o_s| \geq j - i + 1 = \lfloor D/2 \rfloor + 1 > D/2$. If $j = i + \lfloor D/2 \rfloor - D$, then $|N^o_s| \geq i - j + 1 = D - \lfloor D/2 \rfloor + 1 > D/2$. In both cases we have a contradiction, and hence function π satisfies property 3.1.

The notion of *capture* can be used to construct a bipartite graph $H = (S, O, E)$ (Figure 5). For each facility in S we have a vertex on the S-side, and for each facility in O we have a vertex on the O-side. We add an edge between $s \in S$ and $o \in O$ if s captures o. It is easy to see that each vertex on the O-side has degree at most one, while vertices on the S-side can have degree up to k. We call H the *capture graph*.

We now consider k swaps, one for each facility in O. If some bad facility $s \in S$

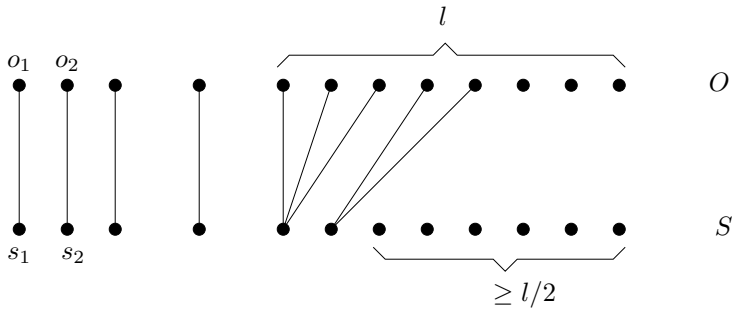


FIG. 5. *Capture graph* $H = (S, O, E)$.

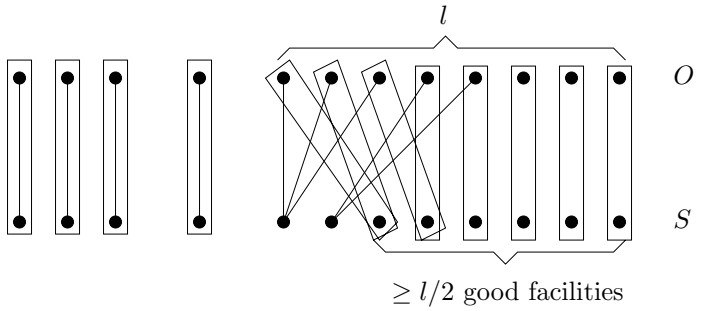


FIG. 6. *k swaps considered in the analysis.*

captures exactly one facility $o \in O$, then we consider the swap $\langle s, o\rangle$. Suppose l facilities in S (and hence l facilities in O) are not considered in such swaps. Each facility out of these l facilities in S is either good or captures at least two facilities in O. Hence there are at least $l/2$ good facilities in S. Now, consider l swaps in which the remaining l facilities in O get swapped with the good facilities in S such that each good facility is considered in at most two swaps (Figure 6). The bad facilities which capture at least two facilities in O are not considered in any swaps. The swaps considered above satisfy the following properties.

1. Each $o \in O$ is considered in exactly one swap.
2. A facility $s \in S$ which captures more than one facility in O is not considered in any swap.
3. Each good facility $s \in S$ is considered in at most two swaps.
4. If swap $\langle s, o\rangle$ is considered, then facility s does not capture any facility $o' \neq o$.

We now analyze these swaps one by one. Consider a swap $\langle s, o\rangle$. We place an upper bound on the increase in the cost due to this swap by reassigning the clients in $N_S(s) \cup N_O(o)$ to the facilities in $S - s + o$ as follows (Figure 7). The clients $j \in N_O(o)$ are now assigned to o. Consider a client $j' \in N_s^{o'}$ for $o' \neq o$. As s does not capture o', by Property 3.1 of π we have that $\pi(j') \notin N_S(s)$. Let $\pi(j') \in N_S(s')$. Note that the distance that the client j' travels to the nearest facility in $S - s + o$ is at most $c_{j's'}$. From triangle inequality, $c_{j's'} \leq c_{j'o'} + c_{\pi(j')o'} + c_{\pi(j')s'} = O_{j'} + O_{\pi(j')} + S_{\pi(j')}$. The clients which do not belong to $N_S(s) \cup N_O(o)$ continue to be served by the same facility. From inequality (1) we have

$$cost(S - s + o) - cost(S) \geq 0.$$

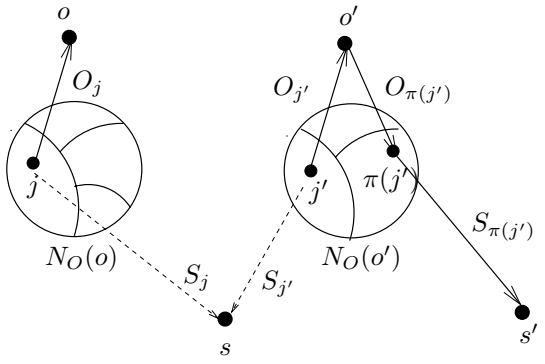


FIG. 7. *Reassigning the clients in* $N_S(s) \cup N_O(o)$.

Therefore,

$$\sum_{j \in N_O(o)} (O_j - S_j) + \sum_{\substack{j \in N_S(s), \\ j \notin N_O(o)}} (O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) \geq 0. \tag{2}$$

As each facility $o \in O$ is considered in exactly one swap, the first term of inequality (2) added over all k swaps gives exactly $cost(O) - cost(S)$. For the second term, we will use the fact that each $s \in S$ is considered in at most two swaps. Since S_j is the shortest distance from client j to a facility in S, using triangle inequality we get $O_j + O_{\pi(j)} + S_{\pi(j)} \geq S_j$. Thus the second term of inequality (2) added over all k swaps is no greater than $2\sum_{j \in C}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j)$. But since π is a 1-1 and onto mapping, $\sum_{j \in C} O_j = \sum_{j \in C} O_{\pi(j)} = cost(O)$ and $\sum_{j \in C}(S_{\pi(j)} - S_j) = 0$. Thus, $2\sum_{j \in C}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) = 4 \cdot cost(O)$. Combining the two terms, we get $cost(O) - cost(S) + 4 \cdot cost(O) \geq 0$. Thus we have the following theorem.

THEOREM 3.2. *A local search procedure for the metric k-median problem with the local neighborhood structure defined by* $\mathcal{B}(S) = \{S - \{s\} + \{s'\} \mid s \in S\}$ *has a locality gap of at most* 5.

The above algorithm and analysis extend very simply to the case when the clients $j \in C$ have arbitrary demands $d_j \geq 0$ to be served.

3.3. Local search with multiswaps. In this section, we generalize the algorithm in section 3 to consider multiswaps in which up to $p > 1$ facilities could be swapped simultaneously. The neighborhood structure is now defined by

$$\mathcal{B}(S) = \{(S \setminus A) \cup B \mid A \subseteq S,\ B \subseteq F, \text{ and } |A| = |B| \leq p\}. \tag{3}$$

The neighborhood captures the set of solutions obtainable by deleting a set of at most p facilities A and adding a set of facilities B where $|B| = |A|$; this swap will be denoted by $\langle A, B \rangle$. We prove that the locality gap of the k-median problem with respect to this operation is exactly $(3 + 2/p)$.

3.4. Analysis. We extend the notion of capture as follows. For a subset $A \subseteq S$, we define

$$capture(A) = \{o \in O \mid |N_S(A) \cap N_O(o)| > |N_O(o)|/2\}.$$

It is easy to observe that if $X, Y \subseteq S$ are disjoint, then $capture(X)$ and $capture(Y)$ are disjoint and if $X \subset Y$, then $capture(X) \subseteq capture(Y)$. We now partition S into sets $A_1, \ldots, A_r$ and O into sets $B_1, \ldots, B_r$ such that

procedure Partition;
  $i = 0$
  while $\exists$ a bad facility in S **do**
    1. $i = i + 1$ {iteration i}
    2. $A_i \leftarrow \{b\}$ where $b \in S$ is any bad facility
    3. $B_i \leftarrow capture(A_i)$
    4. while $|A_i| \neq |B_i|$ **do**
      4.1. $A_i \leftarrow A_i \cup \{g\}$ where $g \in S \setminus A_i$ is any good facility
      4.2. $B_i \leftarrow capture(A_i)$
    5. $S \leftarrow S \setminus A_i$
      $O \leftarrow O \setminus B_i$
  $A_r \leftarrow S$
  $B_r \leftarrow O$
end.

FIG. 8. *A procedure to define the partitions.*

1. for $1 \leq i \leq r-1$, we have $|A_i| = |B_i|$ and $B_i = capture(A_i)$; since $|S| = |O|$, it follows that $|A_r| = |B_r|$;
2. for $1 \leq i \leq r-1$, the set A_i has exactly one bad facility;
3. the set A_r contains only good facilities.

A procedure to obtain such a partition is given in Figure 8.

CLAIM 3.1. *The procedure defined in Figure* 8 *terminates with partitions of S and O, satisfying the properties listed above.*

Proof. The condition in the while loop in step 4 and the assignment in step 5 of the procedure maintain the invariant that $|S| = |O|$. Steps 3 and 4.2 of the procedure ensure that for $1 \leq i \leq r-1$, we have $B_i = capture(A_i)$, and steps 2 and 4.1 ensure that each for $1 \leq i \leq r-1$, the set A_i has exactly one bad facility. Now before each execution of step 4.1, we have $|A_i| < |B_i|$. This together with the invariant that $|S| = |O|$ implies that in step 4.1, we can always find a good facility in $S \setminus A_i$. Since with each execution of the while loop in step 4 the size of A_i increases, the loop terminates. The condition in step 4 then ensures that for $1 \leq i \leq r-1$, we have $|A_i| = |B_i|$. Since there are no bad facilities left when the procedure comes out of the outer while loop, we have that the set A_r contains only good facilities. □

We now use this partition of S and O to define the swaps we would consider for our analysis. We also associate a positive real weight with each such swap.

1. If $|A_i| = |B_i| \leq p$ for some $1 \leq i \leq r$, then we consider the swap $\langle A_i, B_i \rangle$ with weight 1. From the local optimality of S we have

$$cost((S \setminus A_i) \cup B_i) - cost(S) \geq 0.$$

   Note that even if $A_i \cap B_i \neq \emptyset$ or $S \cap B_i \neq \emptyset$, the above inequality continues to hold.
2. If $|A_i| = |B_i| = q > p$, we consider all possible swaps $\langle s, o \rangle$ where $s \in A_i$ is a good facility and $o \in B_i$. Note that if $i \neq r$, there are exactly $q-1$ good facilities in A_i, and for $i = r$ we select any $q-1$ out of the q good facilities in A_r. We associate a weight of $1/(q-1)$ with each of these $q(q-1)$ swaps. For each such swap $\langle s, o \rangle$, we have

$$cost(S - s + o) - cost(S) \geq 0.$$

Note that any good facility in A_i is considered in swaps of total weight at most $q/(q-1) \leq (p+1)/p$. The swaps we have considered and the weights we assigned to them satisfy the following properties:

1. For every facility $o \in O$, the sum of weights of the swaps $\langle A, B\rangle$ with $o \in B$ is exactly one.
2. For every facility $s \in S$, the sum of weights of the swaps $\langle A, B\rangle$ with $s \in A$ is at most $(p+1)/p$.
3. If a swap $\langle A, B\rangle$ is considered, then $capture(A) \subseteq B$.

For each facility $o \in O$, we partition $N_O(o)$ as follows:

1. For $|A_i| \leq p, 1 \leq i \leq r$, let $N^o_{A_i} = N_S(A_i) \cap N_O(o)$ be a set in the partition.
2. For $|A_i| > p, 1 \leq i \leq r$, and all $s \in A_i$, let $N^o_s = N_S(s) \cap N_O(o)$ be a set in the partition.

As before, for each facility $o \in O$, we consider a 1-1 and onto mapping $\pi : N_O(o) \to N_O(o)$ with the following property.

PROPERTY 3.2. *For all sets P, in the partition of $N_O(o)$ for which $|P| \leq \frac{1}{2}|N_O(o)|$, we have $\pi(P) \cap P = \emptyset$.* Such a mapping π can be defined in a manner identical to the one described in section 3.2. The analysis is similar to the one presented for the single-swap heuristic. For each of the swaps defined above, we upper-bound the increase in the cost by reassigning the clients. Property 3.2 ensures that the function π can be used to do the reassignment as described in section 3.2. We take a weighted sum of the inequalities corresponding to each of the swaps considered above. Recall that in the single-swap analysis, we used the fact that each facility in S was considered in at most two swaps and upper-bounded the second term of (2) by $2\sum_{j \in C}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) = 4 \cdot cost(O)$. Similarly, we can now make use of the fact that each facility in S is considered in swaps with total weight at most $(p+1)/p$ and upper-bound the second term by $(p+1)/p \cdot \sum_{j \in C}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) = 2(p+1)/p \cdot cost(O)$. This gives us a locality gap of $1 + 2(p+1)/p = 3 + 2/p$.

3.5. Tight example. In Figure 9, we show an instance of the k-median problem in which a solution that is locally optimum for the 2-swap heuristic ($p = 2$) has cost at least $4 - o(1)$ times the cost of the global optimum. Since $3 + 2/p = 3 + 2/2 = 4$ is also the locality gap proved, it shows that the analysis of the 2-swap heuristic is tight. This tight example can be generalized for p-swaps for any $p \geq 1$. In Figure 9, the black squares are the facilities $\{o_1, o_2, \ldots, o_k\}$ opened by a solution O, the gray squares are the facilities $\{s_1, s_2, \ldots, s_k\}$ opened by a locally optimum solution S, and the circles are the clients. In the graph in Figure 9 each edge has a label which is its length. The cost of serving a client j by a facility i is length of the shortest path between client j and facility i in the graph; the cost is infinite if there is no path.

Note that $cost(S) = \frac{8k-10}{3}, cost(O) = \frac{2k+2}{3}$, and hence the ratio $cost(S)/cost(O)$ approaches 4 as k approaches ∞. We now show that S is a locally optimum solution; that is, if we swap $\{o_l, o_m\}$ for $\{s_i, s_j\}$, then the cost does not decrease. To show this we consider various cases:

1. $i, j \leq r$. Then o_l, o_m will have to lie in the connected components containing s_i, s_j. But in this case the cost would increase by 4.
2. $i \leq r < j$. At least one of o_l, o_m would have to lie in the connected component containing s_i; let this be o_l. If o_m also lies in this component, then the cost remains unchanged. If o_m is in a different component and $m \leq k-2$, then the cost increases by 2. If $m > k-2$, then the cost of the solution increases by 3.

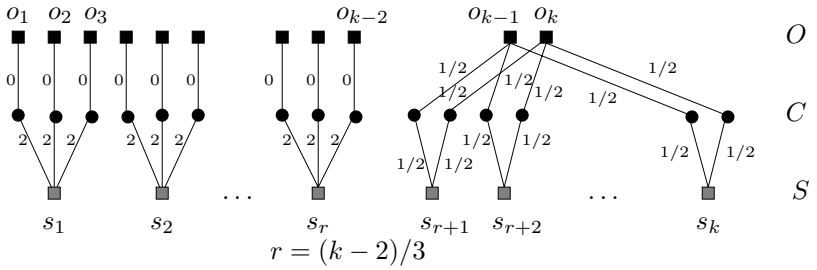


FIG. 9. *Tight example for* 2*-swap. The same example can be generalized to p-swap.*

3. $i, j > r$. If both l, m are at most $k-2$, then the cost of the solution remains unchanged. The cost remains unchanged even if $l \leq k-2 < m$. If both l, m are larger than $k-2$, then, once again, the cost of the solution remains unchanged.

4. Uncapacitated facility location. In facility location problems, we can open any number of facilities, but each facility $i \in F$ has a cost $f_i \geq 0$ of opening it. The UFL problem is to identify a subset $S \subseteq F$ and to serve the clients in C by the facilities in S such that the sum of facility costs and service costs is minimized. That is, if a client $j \in C$ is assigned to a facility $\sigma(j) \in S$, then we want to minimize $cost(S) = \sum_{i \in S} f_i + \sum_{j \in C} c_{\sigma(j)j}$. Note that for a fixed S, serving each client by the nearest facility in S minimizes the service cost.

4.1. A local search procedure. We present a local search procedure for the metric UFL problem with a locality gap of 3. The operations allowed in a local search step are adding a facility, deleting a facility, and swapping facilities. Hence the neighborhood $\mathcal{B}$ is defined by

$$\mathcal{B}(S) = \{S + \{s'\}\} \cup \{S - \{s\} \mid s \in S\} \cup \{S - \{s\} + \{s'\} \mid s \in S\}. \tag{4}$$

As the number of neighbors to be checked at each local search step is polynomial, the algorithm can be run in polynomial time as described earlier.

Charikar and Guha [4] proved a locality gap of 3 for a local search procedure where the operation was of adding a facility and dropping zero or more facilities. Korupolu, Plaxton, and Rajaraman [14] considered the operations of adding, deleting, and swapping a facility but could only prove a locality gap of at most 5.

4.2. The analysis. For any set of facilities $S' \subseteq F$, let $cost_f(S') = \sum_{i \in S'} f_i$ denote the facility cost of the solution S'. Also, let $cost_s(S')$ be the total cost of serving the clients in C by the nearest facilities in S'. The total cost of a solution S' is denoted by $cost(S')$. We use S to denote a locally optimum solution. The following bound on the service cost of S has earlier been proved by Korupolu, Plaxton, and Rajaraman [14].

LEMMA 4.1 (service cost).

$$cost_s(S) \leq cost_f(O) + cost_s(O).$$

Proof. Consider an operation in which a facility $o \in O$ is added. Assign all the clients $N_O(o)$ to o. From the local optimality of S we get $f_o + \sum_{j \in N_O(o)} (O_j - S_j) \geq 0$. Note that even if $o \in S$, this inequality continues to hold. If we add such inequalities for every $o \in O$, we get the desired inequality. ∎

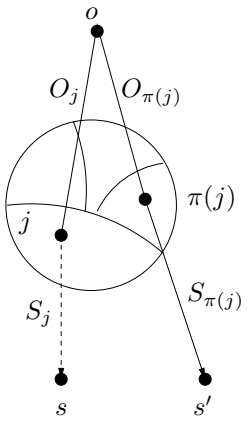


FIG. 10. *Bounding the facility cost of a good facility s.*

The following lemma gives a bound on the facility cost of S.

LEMMA 4.2 (facility cost).

$$cost_f(S) \leq cost_f(O) + 2 \cdot cost_s(O).$$

Proof. As before, we assume that for a fixed $o \in O$, the mapping $\pi : N_O(o) \to N_O(o)$ is 1-1 and onto and satisfies Property 3.1. In addition, we assume that if $|N_s^o| > \frac{1}{2}|N_O(o)|$, then for all $j \in N_s^o$ for which $\pi(j) \in N_s^o$, we have that $\pi(j) = j$. Here we give an outline of how to define such a function π. Let $|N_s^o| > \frac{1}{2}|N_O(o)|$. We pick any $|N_s^o| - |N_O(o) \setminus N_s^o|$ clients j from N_s^o and set $\pi(j) = j$. On the remaining clients in $N_O(o)$, the function π is defined in the same manner as in section 3.2.

Recall that a facility $s \in S$ is *good* if it does not capture any o, that is, for all $o \in O$, $|N_s^o| \leq \frac{1}{2}|N_O(o)|$. The facility cost of good facilities can be bounded easily as follows (see Figure 10). Consider an operation in which a good facility $s \in S$ is dropped. Let $j \in N_S(s)$ and $\pi(j) \in N_S(s')$. As s does not capture any facility $o \in O$, we have that $s' \neq s$. If we assign j to s', then we have $-f_s + \sum_{j \in N_S(s)}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) \geq 0$. Since for all $j \in N_S(s), \pi(j) \neq j$, the term $\sum_{\substack{j \in N_S(s) \\ \pi(j)=j}} O_j$ is trivially zero and hence we can rewrite the above inequality as

$$-f_s + \sum_{\substack{j \in N_S(s) \\ \pi(j) \neq j}} (O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) + 2 \sum_{\substack{j \in N_S(s) \\ \pi(j)=j}} O_j \geq 0. \tag{5}$$

For bounding the facility cost of a bad facility $s \in S$ we proceed as follows. Fix a bad facility $s \in S$. Suppose s captures the facilities $P \subseteq O$. Let $o \in P$ be the facility nearest to s. We consider the swap $\langle s, o \rangle$. The clients $j \in N_S(s)$ are now assigned to the facilities in $S - s + o$ as follows:

1. Suppose $\pi(j) \in N_S(s')$ for $s' \neq s$. Then j is assigned to s'. Let $j \in N_O(o')$. We have, $c_{js'} \leq c_{jo'} + c_{\pi(j)o'} + c_{\pi(j)s'} = O_j + O_{\pi(j)} + S_{\pi(j)}$ (Figure 11).
2. Suppose $\pi(j) = j \in N_S(s)$ and $j \in N_O(o)$. Then j is assigned to o.
3. Suppose $\pi(j) = j \in N_S(s)$ and $j \in N_O(o')$ for $o' \neq o$. By Property 3.1 of the mapping π, facility s captures facility o' and hence $o' \in P$. The client j is now assigned to facility o. From triangle inequality, $c_{jo} \leq c_{js} + c_{so}$. Since $o \in P$ is the closest facility to s, we have $c_{so} \leq c_{so'} \leq c_{js} + c_{jo'}$. Therefore, $c_{jo} \leq c_{js} + c_{js} + c_{jo'} = S_j + S_j + O_j$ (Figure 11).

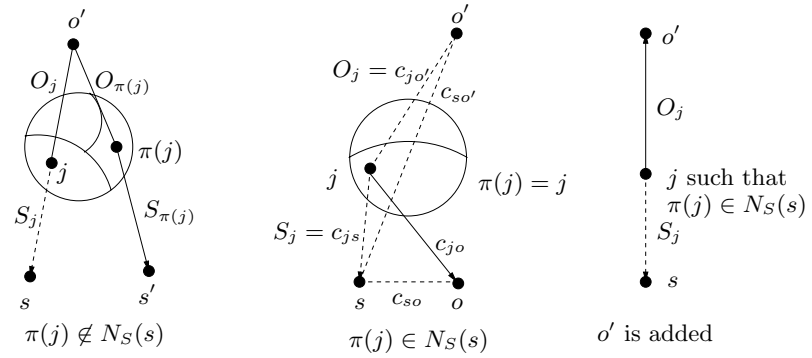


FIG. 11. *Bounding the facility cost of a bad facility s. The figures on the left and in the middle show reassignment when s is dropped, and the figure on the right shows reassignment when o' is added.*

Thus for the swap $\langle s, o\rangle$ we get the following inequality:

$$f_o - f_s + \sum_{\substack{j \in N_S(s) \\ \pi(j) \neq j}} (O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) + \sum_{\substack{j \in N_O(o), \\ \pi(j) = j \in N_S(s)}} (O_j - S_j) + \sum_{\substack{j \notin N_O(o), \\ \pi(j) = j \in N_S(s)}} (S_j + S_j + O_j - S_j) \geq 0. \tag{6}$$

Now consider an operation in which a facility $o' \in P - o$ is added (Figure 11). The clients $j \in N_O(o')$ for which $\pi(j) = j \in N_S(s)$ are now assigned to the facility o', and this yields the following inequality.

$$f_{o'} + \sum_{\substack{j \in N_O(o') \\ \pi(j) = j \in N_S(s)}} (O_j - S_j) \geq 0 \qquad \text{for each } o' \in P - o. \tag{7}$$

Adding inequality (6) to inequalities (7) we get, for a bad facility $s \in S$,

$$\sum_{o' \in P} f_{o'} - f_s + \sum_{\substack{j \in N_S(s), \\ \pi(j) \neq j}} (O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) + 2 \sum_{\substack{j \in N_S(s), \\ \pi(j) = j}} O_j \geq 0. \tag{8}$$

The last term on the left is an upper bound on the sum of the last two terms on the left of inequality (6) and the last term on the left of the inequality (7) added for all $o' \in P - o$.

Now we add inequalities (5) for all good facilities $s \in S$, inequalities (8) for all bad facilities $s \in S$, and inequalities $f_o \geq 0$ for all $o \in O$, which are not captured by any $s \in S$, to obtain

$$\sum_{o \in O} f_o - \sum_{s \in S} f_s + \sum_{\pi(j) \neq j} (O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) + 2 \sum_{\pi(j) = j} O_j \geq 0.$$

Note that $\sum_{j:\pi(j)\neq j} O_j = \sum_{j:\pi(j)\neq j} O_{\pi(j)}$ and $\sum_{j:\pi(j)\neq j} S_j = \sum_{j:\pi(j)\neq j} S_{\pi(j)}$. Therefore we have $\sum_{j:\pi(j)\neq j}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) = 2\sum_{j:\pi(j)\neq j} O_j$ and hence $cost_f(O) - cost_f(S) + 2 \cdot cost_s(O) \geq 0$. This proves the desired lemma. □

Combining Lemmas 4.1 and 4.2, we get the following result.

THEOREM 4.3. *The local search procedure for the metric UFL problem with the neighborhood structure* $\mathcal{B}$ *given by* $\mathcal{B}(S) = \{S + \{s'\}\} \cup \{S - \{s\} \mid s \in S\} \cup \{S - \{s\} + \{s'\} \mid s \in S\}$ *has a locality gap of at most* 3.

The algorithm described above extends very simply to the case when the clients $j \in C$ have arbitrary demands $d_j \geq 0$ to be served. We now show how to use a technique from [4] to obtain $1 + \sqrt{2} + \epsilon \approx 2.414 + \epsilon$ approximation to the UFL. The main idea is to exploit the asymmetry in the service and facility cost guarantees.

Note that Lemmas 4.1 and 4.2 hold for any solution O and not just the optimal solution. We multiply the facility costs by a suitable factor $\alpha > 0$ and solve the new instance using local search.

THEOREM 4.4. *The metric UFL problem can be approximated to factor* $1+\sqrt{2}+\epsilon$ *using a local search procedure.*

Proof. As before, we denote the facility cost and the service cost of an optimum solution O by $cost_f(O)$ and $cost_s(O)$, respectively. Let $cost'_f(A)$ and $cost'_s(A)$ denote the facility and service costs of a solution A in the scaled instance and let S be a locally optimum solution. Then

$$\begin{aligned} cost_f(S) + cost_s(S) &= \frac{cost'_f(S)}{\alpha} + cost'_s(S) \\ &\leq \frac{cost'_f(O) + 2cost'_s(O)}{\alpha} + cost'_f(O) + cost'_s(O) \\ &= (1+\alpha)cost_f(O) + \left(1 + \frac{2}{\alpha}\right) cost_s(O). \end{aligned}$$

The inequality follows from Lemmas 4.1 and 4.2. Now, by setting $\alpha = \sqrt{2}$, we get $cost(S) \leq (1 + \sqrt{2})cost(O)$. Thus local search can be used to obtain a $1 + \sqrt{2} + \epsilon$ approximation. ∎

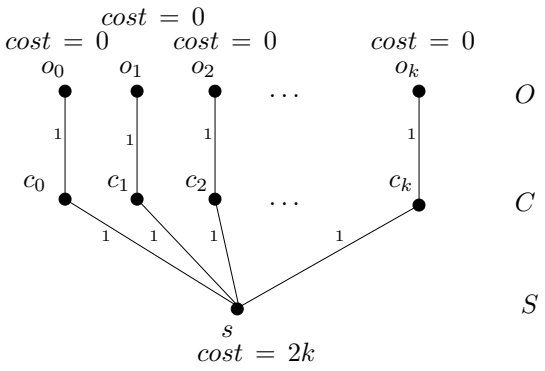


FIG. 12. *Tight example for the locality gap of UFL.*

4.3. Tight example. In Figure 12, we show an instance where a local optimum has cost at least $3 - o(1)$ times the cost of the global optimum. The locally optimum solution S consists of a single facility s while the optimum solution O consists of facilities $\{o_0, o_1, \ldots, o_k\}$. All edges shown have unit lengths, and the cost of serving client j by facility f is the length of the shortest path between client j and facility f in the graph. The cost of opening facility s is $2k$, while that of opening any other facility is zero. To argue that the solution S is locally optimum, note that we cannot delete

facility s. It is also easy to verify that we cannot decrease the cost of our solution by either the addition of any facility from O or by any swap which involves bringing in a facility from O and deleting s. Thus S is locally optimum and has cost $3k+1$, while the cost of O is $k+1$. Since the ratio $cost(S)/cost(O)$ tends to 3 as k tends to ∞, our analysis of the local search algorithm is tight.

5. The capacitated facility location problem. In the CFL problem, along with the facility costs $f_i \geq 0$, we are given integer capacities $u_i > 0$ for each $i \in F$. We can open multiple copies of a facility i. Each copy incurs a cost f_i and is capable of serving at most u_i clients. Note that the capacities u_i may be *different* for different facilities i. The problem is to identify a multiset S of facilities and to serve the clients in C by the facilities in S such that the capacity constraints are satisfied and the sum of the facility costs and the service costs is minimized. Since the clients have unit demands and the facilities have integer capacities, every client will get assigned to a single facility. If a client $j \in C$ is assigned to a facility $\sigma(j) \in S$, then we want to minimize $cost(S) = \sum_{i \in S} f_i + \sum_{j \in C} c_{j\sigma(j)}$. Given an S, the service cost can be minimized by solving a min-cost assignment problem.

In the remainder of this section we let S and O be the multisets of the facilities opened in the locally optimum solution and an optimum solution, respectively.

5.1. A local search algorithm. In this section, we prove a locality gap of at most 4 on a local search procedure for the CFL problem. The operations allowed at each local search step are adding a single copy of a facility $s' \in F$ or adding $l \geq 1$ copies of a facility $s' \in F$ and dropping a subset of the open facilities, $T \subseteq S$. For the second operation l should be sufficiently large so that the clients $j \in N_S(T)$ can be served by these new copies of s', that is, $l \cdot u_{s'} \geq |N_S(T)|$. So the neighborhood structure $\mathcal{B}$ is defined by

$$\mathcal{B}(S) = \{S + s' \mid s' \in F\} \cup \{S - T + l \cdot \{s'\} \mid s' \in F, T \subseteq S, l \cdot u_{s'} \geq |N_S(T)|\}, \tag{9}$$

where $l \cdot \{s'\}$ represents l new copies of s'. If we service all clients in $N_S(T)$ by the new copies of facility s', the cost of the new solution is at most

$$cost(S) + l \cdot f_{s'} + \sum_{s \in T} \left(-f_s + \sum_{j \in N_S(s)} (c_{js'} - c_{js}) \right).$$

Given a facility $s' \in F$, we use the procedure `T-hunt` described in Figure 13 to find a subset, $T \subseteq S$, of facilities to close. Here $m = |C|$ is an upper bound on the number of new copies of s' that we need to open. Closing a facility $s \in S$ gives an extra $|N_S(s)|$ clients to be served by the new facility s'. A client $j \in N_S(s)$ now travels an extra distance of at most $(c_{s'j} - c_{sj})$. Thus, closing facility s gives a *savings* of $f_s - \sum_{j \in N_S(s)} (c_{s'j} - c_{sj})$. Due to capacity constraints, a copy of s' can serve at most $u_{s'}$ clients. This motivates us to define the following `Knapsack` problem. For a facility $s \in S$, define $\texttt{weight}(s) = |N_S(s)|$ and $\texttt{profit}(s) = f_s - \sum_{j \in N_S(s)} (c_{s'j} - c_{sj})$. The oracle $\texttt{Knapsack}(W)$ returns a multiset $T \subseteq S$ such that $\sum_{s \in T} \texttt{weight}(s) \leq W$ and $\texttt{profit}(T) = \sum_{s \in T} \texttt{profit}(s)$ is maximized.

It is interesting to note that since we permit any subset of facilities T from our current solution S to be dropped, the number of operations is exponential in $|S|$. However, by counting the change in cost due to each such operation in a specific way, we are able to give a polynomial time procedure (the procedure `T-hunt`) to identify a local operation which improves the cost. It might be the case that `T-hunt` is not able

Procedure `T-Hunt`.

1. For $l = 1$ to m do,
2. $\quad T \leftarrow \texttt{Knapsack}(l \cdot u_{s'})$.
3. $\quad$ If $cost(S) + l \cdot f_{s'} - \texttt{profit}(T) < cost(S)$, then return T.
4. return "could not find a solution that reduces the cost."

FIG. 13. *A procedure to find a subset $T \subseteq S$ of facilities.*

to identify a local operation which improves the cost even though such operations exist. However, our analysis will work only with the assumption that `T-hunt` could not find a solution which improves the cost.

5.2. The analysis. As the output S is locally optimum with respect to additions, Lemma 4.1 continues to bound the service cost of S. We restate Lemma 4.1 here.

LEMMA 5.1 (service cost).

$$cost_s(S) \leq cost_f(O) + cost_s(O).$$

LEMMA 5.2. *For any $U \subseteq S$ and any $s' \in F$, we have*

$$\lceil |N_S(U)|/u_{s'} \rceil \cdot f_{s'} + \sum_{s \in U} |N_S(s)| \cdot c_{ss'} \geq \sum_{s \in U} f_s.$$

Proof. The algorithm terminated with the output S. Hence for the solution S and for the facility s', the procedure `T-hunt` must have returned "could not find a solution that reduces the cost." Consider the run of the for-loop for $l = \lceil |N_S(U)|/u_{s'} \rceil$. Since $\sum_{s \in U} \texttt{weight}(s) = N_S(U) \leq l \cdot u_{s'}$, the solution T returned by the knapsack oracle has profit at least as large as `profit`(U). Hence,

$$0 \leq l \cdot f_{s'} - \texttt{profit}(T) \leq l \cdot f_{s'} - \texttt{profit}(U) = l \cdot f_{s'} - \sum_{s \in U} \left(f_s - \sum_{j \in N_S(U)} (c_{js'} - c_{js}) \right).$$

However, by triangle inequality we have $c_{js'} - c_{js} \leq c_{ss'}$. Therefore we have proved the lemma. ∎

We are now ready to bound the facility cost of S.

LEMMA 5.3 (facility cost).

$$cost_f(S) \leq 3 \cdot cost_f(O) + 2 \cdot cost_s(O).$$

To prove the above lemma, we consider a directed graph $G = (V, E)$ with lengths on edges, where

$$V = \{v_s \mid s \in S\} \cup \{w_o \mid o \in O\} \cup \{sink\},$$

$$E = \{(v_s, w_o) \mid s \in S, o \in O\} \cup \{(w_o, sink) \mid o \in O\}.$$

The lengths of (v_s, w_o) and $(w_o, sink)$ are c_{so} and f_o/u_o, respectively (see Figure 14). The cost of routing unit flow along any edge is equal to the length of that edge. We want to simultaneously route $|N_S(s)|$ units of flow from each v_s to the *sink*.

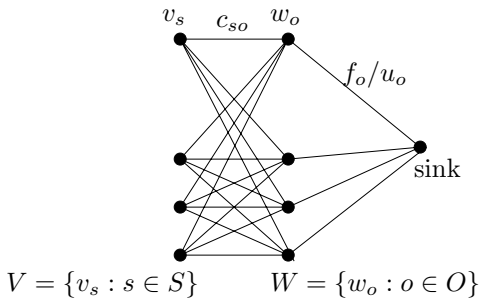


FIG. 14. *The flow graph.*

LEMMA 5.4. *We can simultaneously route $|N_S(s)|$ units of flow from each v_s to the sink such that the total routing cost is at most $cost_s(S) + cost_s(O) + cost_f(O)$.*

Proof. Consider a client $j \in C$. If $j \in N_s^o$, then route one unit of flow along the path $v_s \to w_o \to sink$. Triangle inequality implies $c_{so} \le S_j + O_j$. If for each client we route a unit flow in this manner, then the edge $(w_o, sink)$ carries $N_O(o)$ units of flow at cost $|N_O(o)| \cdot f_o/u_o \le \lceil |N_O(o)|/u_o \rceil \cdot f_o$, which is the contribution of o to $cost_f(O)$. Thus, the routing cost of this flow is at most $cost_s(S) + cost_s(O) + cost_f(O)$. ∎

Since there are no capacities on the edges of graph G, any minimum cost flow must route all $N_S(s)$ units of flow from v_s to the sink along the shortest path. This would be a path (v_s, w_o, sink), where o is such that $c_{so} + f_o/u_o$ is minimized with ties broken arbitrarily. For each $o \in O$, let $T_o \subseteq S$ denote the set of facilities s that route their flow via w_o in this minimum cost flow. From Lemma 5.4, we have

$$cost_s(S) + cost_s(O) + cost_f(O) \ge \sum_{o \in O} \sum_{s \in T_o} |N_S(s)|(c_{so} + f_o/u_o). \tag{10}$$

Now, applying Lemma 5.2 to T_o and o, we get

$$\lceil |N_S(T_o)|/u_o \rceil \cdot f_o + \sum_{s \in T_o} |N_S(s)| \cdot c_{so} \ge \sum_{s \in T_o} f_s.$$

Hence,

$$f_o + |N_S(T_o)|/u_o \cdot f_o + \sum_{s \in T_o} |N_S(s)| \cdot c_{so} \ge \sum_{s \in T_o} f_s.$$

Adding these inequalities for all $o \in O$, we get

$$\sum_{o \in O} f_o + \sum_{o \in O} \sum_{s \in T_o} |N_S(s)|(c_{so} + f_o/u_o) \ge \sum_{o \in O} \sum_{s \in T_o} f_s = cost_f(S). \tag{11}$$

The inequalities (10) and (11) together imply

$$cost_f(S) \le 2 \cdot cost_f(O) + cost_s(O) + cost_s(S).$$

This inequality, together with Lemma 5.1, gives Lemma 5.3. Combining Lemmas 5.1 and 5.3, we obtain the following result.

THEOREM 5.5. *A local search procedure for the metric CFL problem where in each step we can either add a facility or delete a subset of facilities and add multiple copies of a facility has a locality gap of at most* 4.

Using an argument similar to the one in Theorem 4.4 with $\alpha = \sqrt{3} - 1$ we obtain a $2 + \sqrt{3} + \epsilon \approx 3.732 + \epsilon$ approximation. The tight example given in section 4.3 for the UFL problem shows that a locally optimum solution for this problem can have cost three times the cost of the global optimum.

6. Conclusions and open problems. In this paper, we provided tighter analysis of local search procedures for the k-median and UFL problems. Our sharper analysis leads to a $3 + 2/p + \epsilon$ approximation algorithm for the k-median in which there are polynomially many local search steps, each of which can be performed in time $n^{O(p)}$. For CFL, when multiple copies of a facility can be opened, we introduce a new operation and show how a weaker version of this operation can be performed in polynomial time. This leads to a local search procedure with a locality gap of at most 4. We leave open the problem of obtaining tight bounds on the locality gap of this procedure. It would be interesting to identify such operations for other variants of facility location problems.

Acknowledgments. Arya, Garg, Khandekar, and Pandit first published a preliminary version of this paper which did not include the results in sections 3.3 and 3.4. After this version was distributed, Meyerson–Munagala and Arya–Garg–Khandekar–Pandit independently obtained the results in sections 3.3 and 3.4. Garg and Khandekar would like to thank R. Ravi, Amitabh Sinha, and Goran Konjevod for useful discussions.

REFERENCES

[1] A. Archer, R. Rajagopalan, and D. B. Shmoys, *Lagrangian relaxation for the k-median problem: New insights and continuity properties*, in Proceedings of the 11th Annual European Symposium on Algorithms, Springer-Verlag, New York, 2003, pp. 31–42.

[2] E. Arkin and R. Hassin, *On local search for weighted k-set packing*, Math. Oper. Res., 23 (1998), pp. 640–648.

[3] B. Chandra, H. Karloff, and C. Tovey, *New results on the old k-opt algorithm for the traveling salesman problem*, SIAM J. Comput., 28 (1999), pp. 1998–2029.

[4] M. Charikar and S. Guha, *Improved combinatorial algorithms for the facility location and k-median problems*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1999, pp. 378–388.

[5] M. Charikar, S. Guha, E. Tardos, and D. Shmoys, *A constant-factor approximation algorithm for the k-median problem*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 1–10.

[6] F. Chudak, *Improved approximation algorithms for uncapacitated facility location problem*, in Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization, 1998, pp. 182–194.

[7] F. A. Chudak and D. B. Shmoys, *Improved approximation algorithms for a capacitated facility location problem*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 875–876.

[8] F. Chudak and D. Williamson, *Improved approximation algorithms for capacitated facility location problems*, in Proceedings of the 7th Conference on Integer Programming and Combinatorial Optimization, 1999, pp. 99–113.

[9] S. Guha and S. Khuller, *Greedy strikes back: Improved facility location algorithms*, in Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1998, pp. 649–657.

[10] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani, *Greedy facility location algorithms analyzed using dual fitting factor revealing LP*, J. ACM, 50 (2003), pp. 795–824.

[11] K. Jain and V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.

[12] S. Khuller, R. Bhatia, and R. Pless, *On local search and placement of meters in networks*, SIAM J. Comput., 32 (2003), pp. 470–487.

[13] J. Könemann and R. Ravi, *A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 537–546.

[14] M. Korupolu, C. Plaxton, and R. Rajaraman, *Analysis of a Local Search Heuristic for Facility Location Problems*, Technical Report 98-30, DIMACS, Rutgers University, Piscataway, NJ, 1998.

[15] S. Lin and B. Kernighan, *An effective heuristic algorithm for the travelling salesman problem*, Oper. Res., 21 (1973), pp. 498–516.

[16] M. Mahdian and M. Pál, *Universal facility location*, in Proceedings of 11th Annual European Symposium on Algorithms, Springer-Verlag, New York, 2003, pp. 409–422.

[17] M. Mahdian, Y. Ye, and J. Zhang, *Improved approximation algorithms for metric facility location problems*, in Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization, 2002, pp. 229–242.

[18] M. Mahdian, Y. Ye, and J. Zhang, *A 2-approximation algorithm for the soft-capacitated facility location problem*, in Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), 2003.

[19] M. Pál, E. Tardos, and T. Wexler, *Facility location with nonuniform hard capacities*, in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2001, pp. 329–338.

[20] D. Shmoys, E. Tardos, and K. Aardal, *Approximation algorithms for facility location problems*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 265–274.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 563–583

BUFFER OVERFLOW MANAGEMENT IN QoS SWITCHES*

ALEXANDER KESSELMAN†, ZVI LOTKER‡, YISHAY MANSOUR†,
BOAZ PATT-SHAMIR‡, BARUCH SCHIEBER§, AND MAXIM SVIRIDENKO§

Abstract. We consider two types of buffering policies that are used in network switches supporting Quality of Service (QoS). In the *FIFO* type, packets must be transmitted in the order in which they arrive; the constraint in this case is the limited buffer space. In the *bounded-delay* type, each packet has a maximum delay time by which it must be transmitted, or otherwise it is lost. We study the case of overloads resulting in packet loss. In our model, each packet has an intrinsic value, and the goal is to maximize the total value of transmitted packets.

Our main contribution is a thorough investigation of some natural greedy algorithms in various models. For the FIFO model we prove tight bounds on the competitive ratio of the greedy algorithm that discards packets with the lowest value when an overflow occurs. We also prove that the greedy algorithm that drops the earliest packets among all low-value packets is the best greedy algorithm. This algorithm can be as much as 1.5 times better than the tail-drop greedy policy, which drops the latest lowest-value packets.

In the bounded-delay model we show that the competitive ratio of any on-line algorithm for a uniform bounded-delay buffer is bounded away from 1, independent of the delay size. We analyze the greedy algorithm in the general case and in three special cases: delay bound 2, link bandwidth 1, and only two possible packet values.

Finally, we consider the off-line scenario. We give efficient optimal algorithms and study the relation between the bounded-delay and FIFO models in this case.

Key words. buffer overflows, competitive analysis, Quality of Service, FIFO scheduling, deadline scheduling

AMS subject classifications. 90B35, 68M20, 68Q25, 68W01

DOI. 10.1137/S0097539701399666

1. Introduction.

Unlike the "best effort" service provided by the Internet today, next-generation networks will support guaranteed Quality of Service (QoS) features. In order for the network to support QoS each network switch must be able to guarantee a certain level of QoS in some predetermined parameters of interest, including packet loss probability, queuing delay, jitter, and others.

In this work, we consider models based on the IP environment. Implementing QoS in an IP environment is receiving growing attention, since it is widely recognized that future networks would most likely be IP based. There have been a few proposals that address the integration of QoS in the IP framework, and our models are based on some of these proposals, specifically in the general area of providing *differentiated network services* in an IP environment. For example, different customers may get different levels of service, which might depend on the price they pay for the service.

*Received by the editors December 12, 2001; accepted for publication (in revised form) December 11, 2003; published electronically March 23, 2004. An extended abstract of this paper appeared in *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing* (STOC '01), Crete, Greece, 2001.

http://www.siam.org/journals/sicomp/33-3/39966.html

†Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (alx@math.tau.ac.il, mansour@math.tau.ac.il). The research of these authors was partially supported by a grant from the Israel Ministry of Science and Technology.

‡Department of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel (zvilo@eng.tau.ac.il, boaz@eng.tau.ac.il). The research of these authors was partially supported by a grant from the Israel Ministry of Science and Technology.

§IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (sbar@us.ibm.com, sviri@us.ibm.com).

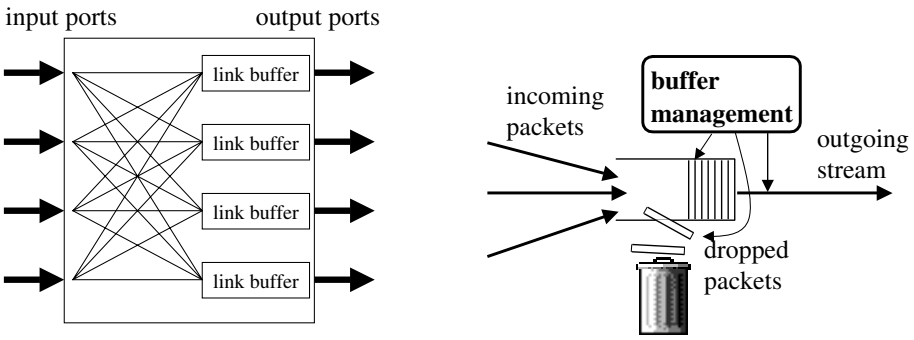


FIG. 1.1. *Schematic representation of the model. Left: general switch structure. Right: output port structure. Packets are placed in the buffer, and the buffer management algorithm controls which packet will be discarded and which will be transmitted.*

One way of guaranteeing QoS is by committing resources to each admitted connection, so that each connection has its dedicated resource set that will guarantee its required level of service regardless of all other connections. This conservative policy (implemented in the specification of CBR traffic in asynchronous transfer mode (ATM) networks [23]) might be extremely wasteful since network traffic tends to be bursty. Specifically, this policy does not take into consideration the fact that *usually*, the worst-case resource requirements of different connections do not occur simultaneously. Recognizing this phenomenon, most modern QoS networks allow some "overbooking," employing the policy popularly known as *statistical multiplexing*. While statistical multiplexing tends to be very cost-effective, it requires satisfactory solutions to the unavoidable events of overload. In this paper we consider such scenarios in the context of *buffering*. The basic situation we consider is an output port of a network switch with the following activities (see Figure 1.1). At each time step, an arbitrary set of packets arrives, but only a fixed number of packets can be transmitted. The buffer management algorithm controls which packets are admitted to the buffer, which are discarded, and which are transmitted at each step.

We consider two types of buffer models. In the *FIFO model*, packets can never be sent out of order: formally, for any two packets p, p' sent at times t, t', respectively, we have that if $t' > t$, then packet p did not arrive after packet p'. The main constraint in this classical model is that the buffer size is fixed, so when too many packets arrive, *buffer overflow* occurs and some packets must be discarded. In most implementations the discard policy is the natural *tail-drop* policy, in which the latest packets are discarded.

The second model we consider is the *bounded delay model*. This model is relatively new, and is warranted by networks that guarantee the QoS parameter of end-to-end delay. Specifically, in the bounded-delay model each packet arrives with a prescribed *allowed delay* time. A packet must be transmitted within its allowed delay time or else it is lost. In this model, the buffer management policy can reorder packets. We consider two variants of the model. In the *uniform* bounded delay model, the switch has a single fixed bound on the delay of all packets, and in the *variable* bounded delay model, the switch may have a different delay bound for each packet.

The focus of our paper is the following simple refinement of the models described above. Each packet arrives with its intrinsic *value*, and the goal of the buffer management algorithm is to discard packets so as to maximize the total value of packets

transmitted. All we assume about the value of the packets is that it is additive; that is, the value of a set of packets is the sum of the values of packets in the set.

In this paper we present a thorough investigation of the natural greedy algorithms in the various models. In the FIFO model, the greedy algorithm discards the lowest-value packets whenever an overflow occurs, with ties broken arbitrarily. We prove a tight bound of $2 - \frac{W}{B+W}$ on the competitive factor of this algorithm, where B is the buffer size and W is the link bandwidth. For the case where the ratio of the maximum to minimum value is bounded by some $\alpha \geq 1$, we prove a tight bound of $2 - \frac{2}{\alpha+1}$ on the competitive factor. The proof of the upper bound is quite involved. We then consider different variants of the greedy algorithms, since the greedy policy does not specify which packet to drop in case there is more than one packet with the lowest value. Specifically, we consider the head-drop greedy policy, which drops the earliest lowest-value packets. We show that for *any* input sequence the head-drop greedy policy achieves equal or better value than any other greedy policy. This is somewhat surprising, since most implementations use the tail-drop policy. Furthermore, we show that the ratio of the value served by the head-drop greedy policy to the value served by the tail-drop policy can be as high as 3/2 in some cases. We also prove a lower bound on the competitive ratio of any on-line algorithm in the FIFO model.

For the bounded delay model we have the following results. First, we show that the competitive ratio of any on-line algorithm for a uniform bounded delay buffer is bounded away from 1, independent of the delay size. This holds even if all packets have an arbitrarily long allowed delay. Next, we consider the simple greedy algorithm, which in this model always sends the packet with the highest value. We prove that the competitive ratio of this algorithm is exactly 2. In the common case when there are only two possible values of packets (i.e., "cheap" and "expensive" packets) the competitive factor of the greedy algorithm is exactly $1 + 1/\alpha$, where $\alpha \geq 1$ is the ratio of the expensive value to the cheap value. We then consider the special case where the delay is less than 2, namely, a packet, if not dropped, is sent when it arrives or in the next time step. We show that in this case, the bound of 2 can be improved: we give algorithms that achieve a competitive ratio of 1.618 for the variable delay model. We also prove lower bounds of 1.11 on the competitive ratio for the uniform delay model and 1.17 for the bounded delay model. Better bounds are presented for the case where the bandwidth link is 1. We show that in this variant slight modifications of the greedy algorithm guarantee better performance.

Lastly, we consider the off-line case. We prove that the overflow management problem has matroid structure in both buffer models, and hence admits efficient optimal off-line algorithms.

Related work. There is a myriad of research papers about packet drop policies in communication networks; see, for example, the survey of [16] and references therein. Some of the drop mechanisms, such as random early detection (RED) [11], are designed to signal congestion to the sending end. The approach abstracted in our model, where each packet has an intrinsic value and the goal is to maximize the total throughput value, is implicit in the recent DiffServ model [8, 9] and ATM [23]. The bounded delay model is an abstraction of the model described in [13].

There has been work on analyzing various aspects of the model using classical queuing theory and assuming Poisson arrivals [21]. The Poisson arrival model has been seriously undermined by the discovery of the heavy tail nature of traffic [18] and the chaotic nature of TCP [24]. In this work we use competitive analysis, which makes no probabilistic assumptions.

The work of [2] concentrated on the case where one cannot discard a packet already in the buffer. The authors give tight bounds on the competitive factor of various algorithms for the special case where there are only two different weights. In [20], the question of video smoothing is studied. One of the results in that paper, which we improve here, is an upper bound of 4 on the competitive ratio of the greedy algorithm for the FIFO model. The work of [14] deals with the loss-competitive analysis rather than the throughput-competitive analysis. The authors show how to translate the loss guarantee to the throughput guarantee and obtain an almost tight upper bound for the case of two packet values. A similar result is presented in this paper for the bounded delay model. The work of [3] studies bandwidth allocation from the competitive analysis viewpoint, disregarding buffer overflows.

The bounded delay model can be viewed as a scheduling problem. In this problem we are given parallel machines (whose number is the link bandwidth) and jobs with release time and deadline that correspond to the packets arriving to the pool. The goal is to maximize the throughput, defined as the sum of the weights of the jobs that terminate by their deadline. In our case we also have the additional constraint that all jobs have the same processing time. The off-line variant of this scheduling problem denoted by $P \,|\, r_i; p_i = p \,|\, \sum w_i(1 - U_i)$ in the standard scheduling notation can be solved in polynomial time using maximum matching. To the best of our knowledge the on-line variant of this problem has not been considered elsewhere. The more general off-line problem, where the processing time is not fixed, is NP-hard. Recently, approximation algorithms for this problem were considered in [5, 4, 22]. The more general on-line problem, where the processing time is not fixed, was considered in [19], where the authors prove competitive ratios that are substantially larger than those in our case. Slightly better ratios can be achieved for the case where processing time is not fixed if preemption is allowed. This case was considered in [6, 7, 15]. For the unweighted version of this problem, i.e., when the goal is to maximize the number of jobs completed by their deadline (which is trivial if all processing times are the same), [6] showed a tight competitive factor of 4. For the weighted version the tight bound is $\sqrt{1 + k^2}$, where k is the ratio of the maximum value density to the minimum value density of a job [15].

Paper organization. The remainder of this paper is organized as follows. In section 2 we define the models and some notation. In section 3 we consider the FIFO model and in section 4 we consider the bounded delay models. Finally in section 5 we discuss off-line algorithms.

2. Model and notation. In this section we formalize the model and the notation we use. We consider two main models: the FIFO model and the bounded-delay switch model. First, we list the assumptions and define quantities that are common to both models.

We assume that time is discrete. Fix an algorithm A. At each time step t, there is a set of *packets* $Q_A(t)$ stored at the *buffer* (initially empty). Each packet p has a positive real *value* denoted by $v(p)$. At time t, a set of packets $A(t)$ arrives. A set of packets from $Q_A(t) \cup A(t)$, denoted by $S_A(t)$, is *transmitted*. We denote by $S_A = \cup_t S_A(t)$ the set of packets transmitted by the algorithm A and by $S_A^{val} = \{p : p \in S_A, v(p) = val\}$ the set of packets of value val sent by A. Note that we consider the so-called cut-through model in which a packet may arrive and be transmitted at the same step. A subset of $Q_A(t) \cup A(t) \setminus S_A(t)$, denoted by $D_A(t)$, is *dropped*. The set of packets in the buffer at time $t + 1$ is $Q_A(t + 1) = Q_A(t) \cup A(t) \setminus (D_A(t) \cup S_A(t))$. We omit subscripts when no confusion arises.

The input is the packet arrival function $A(\cdot)$ and the packet value function $v(\cdot)$. Packets may have other attributes as well, depending on the specific model. The buffer management algorithm has to decide at each step which of the packets to drop and which to transmit, satisfying some constraints specified below. For a given input, the value *served* by the algorithm is the sum of the values of all packets transmitted by the algorithm. For a set P of packets define $v(A)$ to be the total value of the packets in the set. In this notation the value served by algorithm A is $\sum_t v(S_A(t))$.

The sequence of packets transmitted by the algorithm must obey certain restrictions.

Output link bandwidth. We assume that there is an integer number W called the *link bandwidth* such that the algorithm cannot transmit more than W packets in a single time unit; i.e., $|S(t)| \leq W$ for all t. For simplicity, we usually assume that $W = 1$ unless stated otherwise.

FIFO buffers. In the FIFO model there are two additional constraints. First, the sequence of transmitted packets has to be a subsequence of the arriving packets. That is, if a packet p is transmitted after packet p', then p could not have arrived before p'. Second, the number of packets in the buffer is bounded by the *buffer size* parameter, denoted by B. Formally, the constraint is that for all times t, $|Q(t)| \leq B$. (Note that our model allows for a larger number of packets in the transient period of each step, starting with packets' arrival and ending with packets' drop and transmission.)

Bounded-delay buffers. In this model we assume that packets have another attribute: for each packet p there is the *slack time* of p, denoted by $sl(p)$. The requirement is that a packet $p \in A(t)$ must be either transmitted or dropped before time $t + sl(p)$ (i.e., in one of the steps $t, t+1, \ldots, t+sl(p)-1$). The time $t + sl(p)$ is also called the *deadline* of p, denoted by $dl(p)$. We emphasize that in this model there is no explicit bound on the size of the buffer, and that packets may be reordered.

Uniform and variable bounded-delay buffers. One case of special interest within the bounded-delay buffers model is when the slack of all packets is equal to some known parameter δ. We call this model *δ-uniform bounded-delay buffers.* If all packet slacks are only *bounded* by some number δ, we say that the buffer is *δ-variable bounded-delay.*

On-line and off-line algorithms. We call an algorithm *on-line* if for all time steps t, it has to decide which packets to transmit and which to drop at time t without any knowledge of the packets arriving at steps $t' > t$. If future packet arrival is known, the algorithm is called *off-line.* We denote the optimal policy by OPT and the set of packets sent by the optimal policy by S_O. The *competitive ratio* (or *competitive factor*) of an algorithm A is an upper bound, over all input sequences P, on the ratio of the maximal value that can be transmitted by OPT to the value that is transmitted by A, that is, $\max_P(\frac{v(OPT)}{v(A)})$. Note that since we deal with a maximization problem this ratio will always be at least 1. In what follows, we denote by S_O and S_G the sets of packets transmitted by OPT and by the on-line algorithm considered, respectively, if it is not explicitly stated otherwise.

3. The FIFO model. In this section we consider the FIFO model. Recall that in this model a buffer of size B is used to store the incoming packets. Packets have to be transmitted in the order in which they arrive. Each packet has a value associated with it and the goal is to maximize the value of the transmitted packets.

First, we prove a lower bound on the competitive ratio of any on-line algorithm in the FIFO model. This proof improves the 1.25 bound proved in [20].

THEOREM 3.1. *The competitive ratio of any deterministic on-line algorithm in the FIFO model is at least* 1.281.

Proof. Assume that the link rate is 1 packet per time unit. Fix an on-line algorithm A, and let C_A denote its competitive ratio. We consider two scenarios. In both scenarios, B packets of value 1 arrive at time $t = 0$. In each of the next time steps a single packet of value $\alpha > 1$ arrives. This continues until we reach time B or until A sends an α value packet (which means that A has dropped all the remaining packets of value 1 from the buffer). Let t be that time.

In the first scenario, the input stream ends at time t. The benefit of A is $1 \cdot t + \alpha \cdot t$, while the off-line benefit is $1 \cdot B + \alpha \cdot t$.

In the second scenario, at time $t+1$ a burst of B packets, each of value α, arrive. The off-line benefit in this case is $\alpha(B + t)$, while the benefit of A is $1 \cdot t + \alpha \cdot B$.

Thus, the competitive ratio of A is

$$C_A \geq \max\left(\frac{B + \alpha t}{t + \alpha t}, \frac{\alpha(B + t)}{t + \alpha B}\right). \tag{3.1}$$

In our model, the adversary first chooses α so as to maximize C_A, then the algorithm chooses t (possibly depending on α) so as to minimize C_A, and then the competitive ratio is computed from (3.1). We work backwards, assuming first that α is a parameter. It is easy to see that (3.1) is minimized when

$$\frac{B + \alpha t}{t + \alpha t} = \frac{\alpha(B + t)}{t + \alpha B}. \tag{3.2}$$

Solving (3.2) for t, we get that its only nonnegative root is

$$t_0(B, \alpha) = B\frac{\sqrt{(\alpha - 1)^2 + 4\alpha^3} - \alpha + 1}{2\alpha^2}.$$

Define $t_0'(\alpha) = t_0(B, \alpha)/B$. With this notation, it follows from (3.1) that the competitive ratio of A satisfies

$$C_A \geq \frac{1 + \alpha t_0'(\alpha)}{t_0'(\alpha) + \alpha t_0'(\alpha)}. \tag{3.3}$$

Using numerical methods, we find that the right-hand side of (3.3), viewed as a function of α, is maximized when $\alpha \approx 4.01545$. Substituting in (3.1), we obtain that the competitive ratio of A cannot be better than 1.28197. □

3.1. Tight analysis of the greedy algorithm. We consider the greedy algorithm for output link bandwidth W, for any integer $W > 0$. In this algorithm no packets are dropped in time step t in which $|Q(t)|+|A(t)| \leq B+W$. If $|Q(t) \cup A(t)| = k > 0$, then the earliest $\min(W, k)$ packets are transmitted and the rest are stored in the buffer. If $|Q(t)| + |A(t)| = k > B + W$, the $k - B - W$ packets with the lowest value are dropped, ties broken arbitrarily. Among the remaining $B + W$ packets, the W earliest packets are transmitted and the rest are stored in the buffer.

For the sake of simplicity, we assume that B is a multiple of W.[1] Let $B_W = B/W$ denote the number of time steps that is needed to transmit the whole buffer. We first show that the competitive ratio of the greedy algorithm is no worse than 2. Later, we improve this bound and show that the improved bounds are tight.

[1] If W does not divide B, one can think of "fractions" of time steps in which sets of less than W packets are transmitted. This makes the analysis somewhat cumbersome, but the results of this section carry over to the general case as well.

THEOREM 3.2. *The competitive ratio of the greedy algorithm is at most* 2 *for any output link bandwidth* W.

Proof. Consider a sequence of packets. We need to show that $v(OPT) \leq 2v(Greedy)$. Let $D_G^O(t) = S_O \cap D_G(t)$ denote the set of packets that were dropped by the greedy algorithm at time t but were transmitted by the optimal algorithm. Let D_G^O be the union of all these sets. We show a mapping from the packets in D_G^O to packets in S_G with the following properties: (i) a packet from D_G^O is mapped to a packet in S_G with at least the same value, (ii) at most one packet from D_G^O is mapped to any packet in $S_G \cap S_O$, and (iii) at most two packets from D_G^O are mapped to any packet in $S_G \setminus S_O$. Clearly, the existence of the mapping implies the theorem.

We construct the mapping iteratively. At each iteration we consider a packet from D_G^O and map it to a packet in S_G. The packets in D_G^O are considered in the order of their drop time. Suppose that the current packet to be mapped is $p \in D_G^O(t)$. This means that all the packets in $D_G^O(s)$, for $s < t$, and maybe some of the packets in $D_G^O(t)$ have been mapped already. Given the partially defined mapping, define a set of "available" packets in S_G as follows. A packet in $q \in S_G \cap S_O$ is available if so far no packet is mapped to q. A packet in $q \in S_G \setminus S_O$ is available if so far at most one packet is mapped to q. The packet p is mapped to the earliest available packet that was transmitted by the greedy algorithm at or after time t.

Clearly, the mapping defined above satisfies the second and third properties. It remains to be shown is that it satisfies the first property as well. Consider the mapping process step by step, where in each step a single packet is mapped. We prove that the first property holds by induction on the steps of the mapping process. For the basis of the induction note that if $p \in D_G^O(t)$ is the first packet to be dropped, then it is mapped to the packet transmitted by the greedy algorithm at time t. Clearly, the value of this packet is at least $v(p)$.

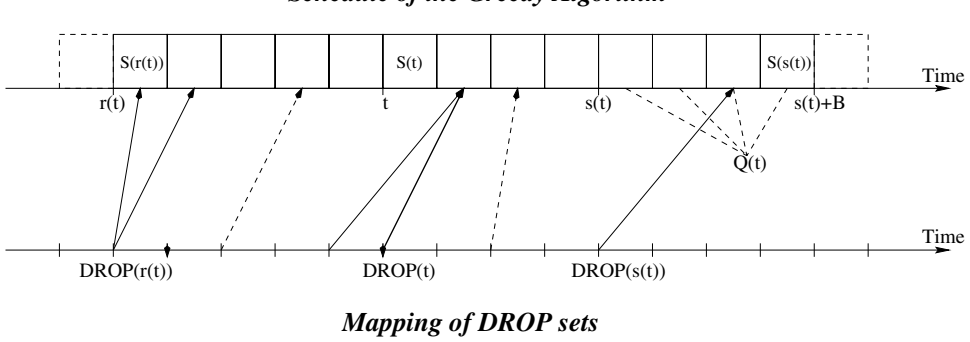


FIG. 3.1. *An example of the mapping from* D_G^O *to* S_G.

Suppose that the first property holds for all packets mapped before $p \in D_G^O(t)$. We show that it also holds for p. Let $s(t)$ be the earliest time at or after t such that the buffer maintained by the greedy algorithm at time $s(t)$ is full and all the packets in the buffer at time $s(t)$ are transmitted by the greedy algorithm (see Figure 3.1). Observe that since the buffer is full at time t, $s(t)$ is well defined.

CLAIM 1. *The value of each of the packets transmitted by Greedy at times* $t, \ldots, s(t) + B_W$ *is at least* $v(p)$.

Proof. Consider the greedy algorithm in the time interval $t, \ldots, s(t) + B_W$. Construct a sequence of time steps inductively as follows:

- $t_0 = t$.
- For $t_i < s(t)$, define t_{i+1} to be the first step after t_i in which a packet that was in the buffer at time t_i is dropped.

Let t_k be the last time step in this sequence. Note that $t_k = s(t_i)$ for all $0 \leq i \leq k$ by definition. We prove the claim by induction on k. For the case $k = 0$ we have that $s(t) = t$, and since all packets in the buffer at time $s(t)$ are transmitted in steps $s(t), \ldots, s(t) + B_W$, and all these packets have value at least $v(p)$ by the greedy rule, we are done. For the induction step, assume that the claim holds for k and consider $k + 1$. Let q be a packet that was in the buffer at time t_0 and dropped at time t_1. By the FIFO ordering, all packets transmitted in steps $t_0, \ldots, t_1$ were in the buffer at time t_0. By the greedy rule we have that (i) all packets transmitted at times $t_0, \ldots, t_1$ have value at least $v(p)$, and (ii) $v(q) \geq v(p)$. The claim follows by combining fact (i) with the induction hypothesis applied to q and time t_1. □

To prove that the first property holds for p, we show that it is mapped to one of the packets transmitted at times $t, \ldots, s(t) + B_W$. For this we need to show that at least one of the packets transmitted at times $t, \ldots, s(t) + B_W$ remains available at the time p is mapped. The "number of availabilities" at the time packet p is mapped is defined as the maximum number of packets that can still be mapped to the packets transmitted at times $t, \ldots, s(t) + B_W$. Specifically, every available packet (at the time p is mapped) among the packets transmitted at times $t, \ldots, s(t) + B_W$ that is also in S_O contributes 1 to the number of availabilities. Every available packet (at the time p is mapped) among the packets transmitted at times $t, \ldots, s(t) + B_W$ that is not in S_O contributes 2 to the number of availabilities if it is not mapped and contributes 1 if one packet has been mapped to it already.

Claim 2. *The number of availabilities before any of the packets dropped at time t is mapped is at least $\sum_{i=t}^{s(t)} |D_G^O(i)|$.*

Proof. Let $r(t)$ be the latest time before or at t such that no packets dropped by the greedy algorithm at time $r(t) - 1$ or earlier are mapped to packets transmitted at time $r(t)$ or later. If no such $r(t)$ exists, then define $r(t) = 0$ (the first step).

Intuitively, we start by showing that each interval $r(t), \ldots, s(t) + B_W$ is independent of the other intervals. Specifically, we claim that the number of availabilities before any of the packets dropped at time $r(t)$ and later is mapped is at least $\sum_{i=r(t)}^{s(t)} |D_G^O(i)|$. To see that, let $C(t)$ denote the set of packets available to the greedy algorithm at times $r(t), \ldots, s(t)$ that were transmitted by the optimal algorithm. In other words, $C(t)$ is the subset of packets transmitted by the optimal algorithm that were considered (stored at the beginning of the interval or arrived during the interval) by the greedy algorithm at times $r(t), \ldots, s(t)$. Note that $|C(t)| \leq W \cdot (s(t) - r(t) + 2B_W + 1)$, because packets in $C(t)$ cannot be transmitted by the optimal algorithm in time steps other than $r(t) - B_W, \ldots, s(t) + B_W$.

Next, let $x(t)$ be the number of packets of $C(t)$ that are transmitted by the greedy algorithm at times $r(t), \ldots, s(t) + B_W$. Observe that no packet from S_O that arrives at times $s(t), \ldots, s(t) + B_W$ is transmitted by the greedy algorithm at times $s(t), \ldots, s(t)+B_W$. This is because by definition, all packets stored by Greedy at time $s(t)$ are transmitted, a process that lasts B_W time units. It follows that exactly $x(t)$ packets from $C(t)$ are transmitted by the greedy algorithm at times $r(t), \ldots, s(t)+B_W$. This implies that the number of availabilities before any of the packets dropped at time $r(t)$ and later is mapped is at least $2W \cdot (s(t) + B_W - r(t) + 1) - x(t)$. This is true since by definition of $r(t)$, none of the packets that are dropped before time $r(t)$

is mapped to packets transmitted at times $r(t), \ldots, s(t) + B_W$. It follows that

$$\sum_{i=r(t)}^{s(t)} |D_G^O(i)| \leq W(s(t) - r(t) + 2B_W + 1) - x(t)$$

$$\leq 2W(s(t) + B_W - r(t) + 1) - x(t). \tag{3.4}$$

We can now prove the claim. Since by (3.4) the maximum number of packets that can be mapped to the packets transmitted at times $r(t), \ldots, s(t) + B_W$ is at least $\sum_{i=r(t)}^{s(t)} |D_G^O(i)|$, we have, by the definition of $r(t)$, that the maximum number of packets that can be mapped to the packets transmitted at times $r(t), \ldots, t-1$ is strictly less than $\sum_{i=r(t)}^{t-1} |D_G^O(i)|$. We conclude that the number of availabilities before the packets dropped at t are mapped is at least $\sum_{i=t}^{s(t)} |D_G^O(i)|$. □

The theorem follows directly from Claims 1 and 2. □

We now refine the analysis above to get tighter bounds. Let α be the ratio of the largest value of a packet to the smallest value of a packet (clearly we can assume without loss of generality that all packets have positive values).

THEOREM 3.3. *The competitive ratio of the greedy algorithm is at most* $2(1-\frac{1}{\alpha+1})$ *for any output link bandwidth* W*, where* $\alpha \stackrel{\text{def}}{=} \frac{\max_p\{v_p\}}{\min_p\{v_p\}}$.

Proof. Given the mapping defined above, partition S_G into three subsets as follows.

- G_1 is the set of packets in S_G that are not in S_O and that are not the image of (i.e., not mapped to by) any packet in D_G^O.
- G_2 is the set of packets in S_G that are unavailable after the mapping is done, namely, all packets in $S_G \cap S_O$ that are the image of one packet in D_G^O, and all packets in $S_G \setminus S_O$ that are the image of two packets in D_G^O.
- $G_3 = S_G \setminus (G_1 \cup G_2)$, that is, the packets in $S_G \cap S_O$ that are not the image of any packet in D_G^O, and the packets in $S_G \setminus S_O$ that are the image of one packet in D_G^O.

We associate two packets from S_O with each packet in $q \in G_2$ as follows. If $q \in G_2 \cap S_O$, then the two packets are q itself and the packet mapped to q. If $q \in G_2 \setminus S_O$, then the two packets are the two packets mapped to q. Similarly, we associate a packet from S_O with each packet in $q \in G_3$ as follows. If $q \in G_3 \cap S_O$, then this packet is q. If $q \in G_3 \setminus S_O$, then this packet is the packet mapped to q. Note that this way we associate every packet in S_O with some packet in S_G. Note that $v(q)$ is always at least the value of each of its associated packets, and the fact that no packet is associated with more than two packets implies the bound on the competitive ratio. We are able to improve the bound of 2 on this ratio using the fact that no packet is associated with packets in G_1.

Note that $|S_G| \geq |S_O|$. It follows that $|G_1| \geq |G_2|$, and thus we can match any packet $q \in G_2$ with a mate $p \in G_1$. We move a $\frac{1}{\alpha+1}$ "fraction" of the value of the packets associated with q and associate it with p. Note that after the move the total value associated with q is no more than $2(1-\frac{1}{\alpha+1})v(q)$. Since $v(q) \leq \alpha v(p)$, the total value associated with p is no more than

$$2\frac{1}{\alpha+1}v(q) \leq 2\frac{\alpha}{\alpha+1}v(p) = 2\left(1 - \frac{1}{\alpha+1}\right)v(p). \qquad \square$$

THEOREM 3.4. *The competitive ratio of the greedy algorithm is at most* $2-\frac{W}{B+W}$ *for any output link bandwidth* W.

Proof. Modify the mapping defined in the proof of Theorem 3.2 as follows. For each time t such that $D_G^O(t) \neq \emptyset$, decrease by W the number of availabilities contributed by the packets transmitted by the greedy algorithm at time t. Specifically, if $p \in S_O$, then no packet is mapped to p and if $p \notin S_O$, at most one packet is mapped to p.

We need to show that Claim 2 still holds. For this, it suffices to show that the number of availabilities before any of the packets dropped at time $r(t)$ and later is mapped is at least $\sum_{i=r(t)}^{s(t)} |D_G^O(i)|$. Notice that the number of availabilities is decreased by at most $W(s(t) - r(t) + 1)$. Claim 2 follows for the modified mapping since

$$\begin{aligned}\sum_{i=r(t)}^{s(t)} |D_G^O(i)| &\leq W(s(t) - r(t) + 2B_W + 1) - x(t) \\ &\leq 2W(s(t) + B_W - r(t) + 1) - x(t) - W(s(t) - r(t) + 1).\end{aligned}$$

Notice that the value of the packet transmitted at time t is at least the minimal value of the packets in $D_G^O(t)$. We "shift" a $W/(|D_G^O(t)| + W)$ fraction of the value of each of the packets in $D_G^O(t)$ from the packet to which it has been mapped originally and associate it to one of the packets transmitted by the greedy algorithm at time t. Since $|D_G^O(t)| \leq B$ and $|S_G(t)| = W$, we get that each packet transmitted by the greedy algorithm is associated with at most $1 + \frac{B}{B+W} = 2 - \frac{W}{B+W}$ packets of at most the same value from S_O. □

Finally, we observe that the last two bounds are tight. Consider the following scenario. At the first time step, $B + W$ packets of value 1 arrive; W are transmitted and the rest are kept in the buffer. Then, at each time $t \in \{1, \ldots, B_W\}$, W packets of value $\alpha \geq 1$ arrive and are kept in the buffer. At time $B_W + 1$ the buffer contains B packets of value α; at this time another $B + W$ packets of value α arrive. The greedy algorithm transmits only $B + W$ of these packets, while the optimal algorithm transmits $2B + W$ of value α and only W of value 1. We get that the competitive ratio for this scenario is

$$\frac{(2B + W)\alpha + W}{(B + W)(\alpha + 1)} = 2 - \frac{\alpha(W + 1) + 2B}{(B + W)(\alpha + 1)}.$$

Letting α go to infinity we get the ratio $2 - \frac{W}{B+W}$, and letting B go to infinity we get the ratio $2 - \frac{2}{\alpha+1}$.

3.2. The best greedy algorithm. The greedy algorithm is underspecified: when there are several packets with the same low value, it is not specified which of them is discarded by the greedy algorithm. It is easy to see that for any link bandwidth, the number of packets transmitted is the same for all variants of the greedy algorithm. However, is there a difference between the variants of the greedy algorithm in terms of weighted throughput? In this section we show that, perhaps surprisingly, it is always better to discard the earliest packets, i.e., the packets which spent the most time in the buffer. We call this policy *head-drop*, as the algorithm prefers to drop packets from the head of the buffer. Head-drop is in contrast to the common practice of *tail-drop*, where the newest packets are discarded. We show that there exist scenarios in which tail-drop results in significantly more losses than head-drop. We remark that the head-drop policy [17] enjoys additional advantages in the TCP/IP environment, namely, it helps the congestion avoidance mechanism.

The following theorem proves that the head-drop greedy algorithm is, in a certain sense, the best greedy algorithm.

THEOREM 3.5. *Let G be any greedy algorithm, and let GH be the greedy head-drop algorithm. For any input sequence, the total value transmitted by GH is at least the total value transmitted by G.*

Proof. Fix an input sequence. First, observe that the number of packets in the buffer of G and the number in the buffer of GH at time t are equal for all times t. (This follows from an easy induction on time, which shows that for any given arrival sequence and link bandwidth, the size of the queue in each step is the same for all algorithms that drop packets only when an overflow occurs.) To prove the theorem, we define, for each time step t, a 1-1 mapping from $Q_{GH}(t)$ to $Q_G(t)$ such that each packet $p \in Q_{GH}(t)$ is mapped to a packet $q \in Q_G(t)$ with at least the same value; i.e., $v(p) \leq v(q)$.

We claim that the existence of these mappings implies the theorem as follows. For a packet $p \in Q(t)$ define the *rank* of p to be its rank in the sequence of packets in $Q(t)$ ordered in ascending values. The existence of the mappings implies that for each time step t, the value of the packet ranked i in $Q_{GH}(t)$ is at most the value of the packet ranked i in $Q_G(t)$. Since $|D_{GH}(t)| = |D_G(t)|$, and since in any greedy algorithm $D(t)$ consists of the lowest ranked packets in $Q(t)$, it follows that $v(D_{GH}(t)) \leq v(D_G(t))$.

The value served by an algorithm is $\sum_t v(S(t))$; hence we get

$$\begin{aligned}\sum_t v(S_{GH}(t)) &= \sum_t v(A(t)) - \sum_t v(D_{GH}(t)) \\ &\geq \sum_t v(A(t)) - \sum_t v(D_G(t)) \\ &= \sum_t v(S_G(t)).\end{aligned}$$

All that remains is to prove the existence of the mappings. Fix a time step t and consider $Q_G(t)$ and $Q_{GH}(t)$. To construct the mapping, we use the natural concept of "height" of a packet in a buffer: For an algorithm A and a packet $p \in Q_A(t)$, the *height* of p, denoted by $h_A(p,t)$, is 1 plus the number of packets that have to be either transmitted or dropped before p can be transmitted. In other words, $h_A(p,t)$ is the rank of p in the sequence of packets in $Q_A(t)$ ordered by arrival time. We also say that a packet $p \in Q_A(t)$ is said to be *below* (or *above*) a packet $p' \in Q_A(t)$ if $h_A(p,t) < h_A(p',t)$ (or, respectively, $h_A(p,t) > h_A(p',t)$).

We now define the mapping from $Q_{GH}(t)$ to $Q_G(t)$. Each packet in $Q_{GH}(t) \cap Q_G(t)$ is mapped to itself. To complete the mapping, consider the packets in $Q_{GH}(t) \setminus Q_G(t)$ in ascending order of height. Map each such packet p to the packet with the lowest height in $Q_G(t) \setminus Q_{GH}(t)$ that has not been mapped yet. It is not difficult to see that this mapping is indeed 1-1. We need to show that each packet in $Q_{GH}(t) \setminus Q_G(t)$ is mapped to a packet in $Q_G(t) \setminus Q_{GH}(t)$ of at least the same value. For this we prove the following two lemmas.

LEMMA 3.6. *Let $p \in Q_{GH}(t) \setminus Q_G(t)$ for some time t. Then $v(p) \leq v(q)$, for all $q \in Q_G(t)$ satisfying $h_G(q,t) \leq h_{GH}(p,t)$.*

Proof. Let t_0 be the time in which G dropped p (and hence $p \notin Q_G(t_0+1)$). We prove the lemma by induction on $t - t_0$. For the base case $t = t_0 + 1$, we have by the greedy rule that $v(p) \leq v(p')$ for any $p' \in Q_G(t)$. For the inductive step, let $t > t_0+1$. First, note that p arrived before step $t-1$, since otherwise it must have been dropped at step $t-1$, contradicting the assumption that $t > t_0 + 1$. It follows that

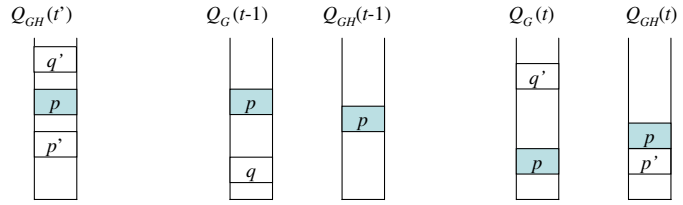


FIG. 3.2. *Scenario considered in the proof of Lemma* 3.7. $p \notin Q_G(t)$, $q' \notin Q_{GH}(t)$, *and* $q \notin D_G(t-1)$.

$h_{GH}(p, t-1)$ is well defined. Consider a packet $p' \in Q_G(t)$ with $h_G(p', t) \leq h_{GH}(p, t)$. If $h_G(p', t-1)$ is defined and $h_G(p', t-1) \leq h_{GH}(p, t-1)$, we are done by induction. If $p' \in A(t-1)$ or if $h_G(p', t-1) > h_{GH}(p, t-1)$, then it must be the case that some packet $p'' \in Q_G(t-1)$ with $h_G(p'', t-1) \leq h_{GH}(p, t-1)$ is dropped by G at time $t-1$. By the greedy rule $v(p') \geq v(p'')$. Since by induction $v(p'') \geq v(p)$, we are done in this case too. □

LEMMA 3.7. *Let t be any time step. If $p \in Q_{GH}(t) \cap Q_G(t)$, then $h_{GH}(p, t) \leq h_G(p, t)$.*

Proof. Suppose that the lemma does not hold. Let t be the first time it is violated, and let p be the packet with the minimal height such that $h_G(p, t) < h_{GH}(p, t)$. Let p' be the packet immediately below p in $Q_{GH}(t)$ (see Figure 3.2). Note that p' is well defined, since by assumption $h_{GH}(p, t) > h_G(p, t) \geq 1$. Also note that $p' \notin Q_G(t)$. This is because otherwise we would also have $h_G(p', t) < h_{GH}(p', t)$, contradicting the height minimality of p. Due to the minimality of t, $h_G(p, t-1) \geq h_{GH}(p, t-1)$. (Note that we cannot have the situation of $p \in A(t-1)$ and $h_G(p, t) < h_{GH}(p, t)$.) Thus, we must have that $D_G(t-1)$ contains at least one packet below p. Denote this packet by q. We claim that $v(q) \geq v(p')$. If $q = p'$, the claim is trivial. Otherwise, we have that $p' \notin Q_G(t-1)$ and $h_{GH}(p', t-1) \geq h_G(q, t-1)$, and hence, by Lemma 3.6, $v(p') \leq v(q)$. Since $Q_G(t)$ has more packets above p than $Q_{GH}(t)$ has, there must be a packet q' above p in $Q_G(t)$ that is not in $Q_{GH}(t)$. Since $q' \notin D_G(t-1)$ and $q \in D_G(t-1)$, it must be that $v(q') \geq v(q) \geq v(p')$. However, the packet q' is not in $Q_{GH}(t)$. Hence, it has been dropped by GH at some $t' < t$. This yields a contradiction to the head-drop rule, since $v(p') \leq v(q')$, both p' and q' are in $Q_{GH}(t')$, and $h_{GH}(q', t') > h_{GH}(p', t')$. □

We can now complete the proof of the theorem by proving the existence of the mapping. Suppose that $p \in Q_{GH}(t) \setminus Q_G(t)$ is mapped to $q \in Q_G(t) \setminus Q_{GH}(t)$. We now show that $h_{GH}(p, t) \geq h_G(q, t)$. By Lemma 3.6 this implies that $v(p) \leq v(q)$. Recall that the mapping of the packets in $Q_{GH}(t) \setminus Q_G(t)$ is done in ascending order of heights and that any such packet is mapped to the packet with the minimal height in $Q_G(t) \setminus Q_{GH}(t)$ that has not been mapped so far. To obtain a contradiction, suppose that p is the packet with the minimal height in $Q_{GH}(t) \setminus Q_G(t)$ that is mapped to $q \in Q_G(t) \setminus Q_{GH}(t)$, and $h_{GH}(p, t) < h_G(q, t)$. Consider the packet $p' \in Q_G(t)$ such that $h_G(p', t) = h_{GH}(p, t)$. It must be that $p' \in Q_G(t) \cap Q_{GH}(t)$. We must also have that the number of packets below p in $Q_{GH}(t) \setminus Q_G(t)$ is the same as the number of packets below p' in $Q_G(t) \setminus Q_{GH}(t)$. Thus, the set of packets below p in $Q_{GH}(t) \cap Q_G(t)$ is the same as the set of packets below p' in $Q_G(t) \cap Q_{GH}(t)$. It follows that $h_G(p', t) < h_{GH}(p', t)$, in contradiction to Lemma 3.7. This completes the proof of Theorem 3.5. □

The following theorem proves that GH can do much better than the greedy tail-

drop algorithm. Let GT denote the greedy tail-drop algorithm.

THEOREM 3.8. *There exist input sequences for which the value transmitted by GH is arbitrarily close to* $3/2$ *times the value transmitted by GT.*

Proof. We describe sequences for the case $W = 1$, using parameters α and B. At time 0, $B/2 + 2$ packets of value 1 arrive. At time 1, $B/2$ packets of value $\alpha \gg 1$ arrive. Thus, at time 2, the head half of the buffer is filled with cheap packets, and the tail half of the buffer is filled with expensive packets. At time 2, $B/2+1$ more 1-value packets arrive (resulting in overflow); finally, at time $2+B/2$, $B+1$ packets of value α arrive, resulting in an additional overflow. Let us now consider the performance of GT and GH. GT drops at time 2 the last $B/2$ cheap packets and transmits a cheap packet in steps $2, 3, \ldots, 1 + B/2$. At time $2 + B/2$, GT drops $B/2$ expensive packets, and the total value eventually transmitted is $\alpha(B+1)+1(\frac{B}{2}+1)$. On the other hand, GH drops at time 2 the first $B/2$ cheap packets and then drops at time $2 + B/2$ the other $B/2$ cheap packets, for a total transmitted value of $\alpha(\frac{3B}{2} + 1) + 1$. The result follows for large B and α values. □

4. Bounded delay buffers. In this section we consider the case of bounded delay buffers. General bounded delay buffers are studied in section 4.1. We prove a lower bound on the competitiveness of any on-line algorithm. The lower bound holds even in the uniform-delay model, independent of the allowed delay. We then show that for the general model, the greedy algorithm is exactly 2-competitive (the bound is refined in case there are exactly two packet values). Finally, in section 4.2 we provide a detailed analysis of the special case where the delay bound is 2.

4.1. General bounded delay buffers. In this section we consider the general case, where the slack times and values of the packets are arbitrary. We first present a lower bound on the competitiveness of all on-line algorithms and then we turn to analyze the simple greedy algorithm. We show that the greedy algorithm is exactly 2-competitive for the general delay-bounded case.

We begin with a negative result motivated by the following (false) intuition. It may seem reasonable to hope that as the delay bound grows, the competitive factor of on-line algorithms might tend to 1, since infinite delay bound seems like the off-line case. This is not true, as proved in the following theorem. The proof is similar in spirit to the proof in the FIFO case (Theorem 3.1).

THEOREM 4.1. *Let* α *be the ratio of the largest to the smallest packet value. Then for any delay bound* δ*, the competitive ratio of any on-line algorithm is at least* $1+\frac{\alpha-1}{\alpha(\alpha+1)}$*, even for uniform-delay buffers. Furthermore, if there are two packet values whose ratio is* $1 + \sqrt{2}$*, then the competitive ratio of any on-line algorithm is at least* $1 + \frac{1}{(1+\sqrt{2})^2} \approx 1.17$ *for any value of* δ*.*

Proof. Let A be any on-line algorithm, and let C_A be its competitive ratio. To bound C_A, consider the following scenario. All packets have the same slack value δ. At time $t = 0$ the buffer is empty and δ packets of value 1 arrive. During each of the following $\delta - 1$ steps ($t = 1, \ldots, \delta - 1$), a single packet of value $\alpha > 1$ arrives. Let x be the number of value 1 packets transmitted by A by time δ. We consider two possible continuations of the scenario. In the first case, no more packets arrive, and in the second case, δ packets of value α arrive at time δ. In the former case, the value served by A is at most $x + \delta\alpha$, while the optimal value is $\delta + \delta\alpha$. In the latter case, the value served by A is $x + \alpha(2\delta - x)$, while the optimal value is $2\delta\alpha$. It follows that the competitive ratio of A is at least

$$C_A \geq \max\left(\frac{\delta(1+\alpha)}{x+\delta\alpha}, \frac{2\delta\alpha}{x+\alpha(2\delta-x)}\right).$$

Consider the two possible ratios. To get the lower bound our goal is to fix α so that the maximum of the two ratios for any value of x is minimized. This is because the on-line algorithm fixes x given the value of α. For any value of α, it is not difficult to see that the best value of x that can be chosen by A is the one where the two ratios are equal. Solving for x as a function of α, we get that the maximum of the ratios is minimized when

$$x(\alpha) = \frac{2\delta\alpha}{\alpha^2+2\alpha-1},$$

in which case the competitive ratio of the algorithm is

$$C_A \geq 1 + \frac{\alpha-1}{\alpha(\alpha+1)}.$$

This proves the first part of the theorem. To prove the second part of the theorem, we find the worst case α by elementary calculus. It turns out that the competitive ratio is maximized (for $x(\alpha)$, i.e., when the algorithm makes the optimal choice) when $\alpha = 1+\sqrt{2}$. In this case we get that $C_A \geq 1 + \frac{1}{(1+\sqrt{2})^2}$, as desired. □

Next, consider the greedy algorithm for the general case where the allowed delays may be different and the link bandwidth is W. The greedy algorithm is extremely simple: at each time step t, transmit the W packets with the highest value whose deadlines have not expired yet. Ties are broken arbitrarily. Note that effectively, the greedy algorithm views each value as a priority class, in the sense that high-priority packets are always transmitted before low-priority ones. For this simplistic strategy, the following relatively strong property was already known [1, 12, 4] in slightly different models.

THEOREM 4.2. *The greedy algorithm is exactly* 2*-competitive in the bounded-delay buffer model, for any output link bandwidth.*

The lower bound for the greedy algorithm holds even if all jobs have the same weight. We note that for the unslotted model (where a packet may arrive during the transmission of another, and preemption is disallowed), [12] proves a lower bound of 2 on the competitive ratio of any deterministic algorithm, and 4/3 for the expected competitive ratio of any randomized algorithms.

In some practical cases, the values assigned to packets are not very refined. In the extreme case, there may be just "cheap" and "expensive" packets, for example, in ATM's Cell Loss Priority bit [23]. We can formalize this model by assigning only two possible values to packets: 1 for "cheap" and $\alpha > 1$ for "expensive." In the following theorem we prove that in this case, the bound guaranteed by Theorem 4.2 can be sharpened to $1 + 1/\alpha$. Notice that the competitive ratio approaches 1 when α tends to infinity.

THEOREM 4.3. *The greedy algorithm is at most* $1 + 1/\alpha$*-competitive in the bounded-delay buffer model with two packet values of* 1 *and* $\alpha > 1$*, for any output link bandwidth.*

Proof. Fix the input sequence, and let W be the output link bandwidth. Consider any optimal algorithm for the sequence. Clearly, $v(S_O^\alpha) \leq v(S_G^\alpha)$. In addition, the greedy algorithm schedules all packets from S_O^1, except the packets that were lost due to the decision of the greedy algorithm to transmit high-value packets with later deadlines. Since each high-value packet may cause loss of at most one low-value packet, we obtain that $v(S_O^1) - v(S_G^1) \leq v(S_G^\alpha)/\alpha$. The theorem follows. □

4.2. The case of $\delta = 2$. Improving on the greedy algorithm in the bounded-delay model turns out to be a challenging task. In this subsection we present a candidate algorithm. However, we are able to analyze its behavior only for the special case of $\delta = 2$, i.e., under the assumption that each packet must be sent either when it arrives or in the following time step. We also present improved lower bounds for this case.

Before we start, let us recall the following well-known fact. A schedule for a set of packets is called *earliest deadline first schedule* (or EDF for short) if the order in which packets are sent is the order of their deadlines.

LEMMA 4.4. *A set of packets with given arrival times and deadlines can be scheduled with link bandwidth W if and only if it can be scheduled by an EDF schedule.*

In addition, packets with the same deadline in a feasible schedule can be further ordered according to their values.

LEMMA 4.5. *A feasible EDF schedule may be transformed into another feasible EDF schedule in which packets with the same deadline are sent in order of nonincreasing value.*

Thus, without loss of generality, we may assume that the optimal algorithm schedules packets in order of nondecreasing deadlines, and packets with the same deadline in order of nonincreasing value. Ties are broken by the arrival order; i.e., packets arriving first are scheduled first.

We use the following algorithms, stated for general delay bound δ and link bandwidth W. The local-EDF algorithm is presented in Figure 4.1. We show in section 5 how the computation of the optimal schedule can be done efficiently. The β-EDF algorithm is defined by a parameter $0 \le \beta \le 1$ and appears in Figure 4.2.

Note that 1-EDF is the greedy algorithm, which sends the W packets with the highest value, and that 0-EDF is the local-EDF algorithm.

THEOREM 4.6. *The $1/\phi$-EDF algorithm is at most ϕ-competitive in the 2-variable bounded-delay buffer model, for any output link bandwidth W.*

Proof. Fix an arrival sequence. We compare the schedule generated by ϕ-EDF with a specific optimal schedule OPT. Specifically, by Lemmas 4.4 and 4.5, we may assume that OPT is EDF, and that if two packets with the same deadline and different values are sent at different times, then the more valuable packet is sent before the less valuable one. We now prove a series of simple properties that follow directly from the definition of the algorithms.

LEMMA 4.7. *If p is transmitted before p' by β-EDF, then p is not transmitted after p' by OPT.*

Proof. Suppose, for contradiction, that there exist packets p, p' such that $p \in S_G(t) \cap S_O(t+1)$ and $p' \in S_O(t) \cap S_G(t+1)$. Clearly both p and p' have deadline $t+1$. The lemma follows from the fact that both β-EDF and OPT send packets with the same deadline in order of nonincreasing value. ∎

LEMMA 4.8. *If $p \in S_G(t)$ performed a push-out at time t, then $v(q') \le v(p)/\phi$ for all packets q' pushed out at time t.*

Proof. It follows from the fact that at Step 3 of the β-EDF, we consider packets pushing out and packets to be pushed out in order of nonincreasing and nondecreasing value, respectively. ∎

LEMMA 4.9. *Suppose that $p, p' \in S_G(t)$ are packets with deadline $t+1$ such that p performed push-out at time t and p' did not perform a push-out at time t. Then $v(p) \le v(p')$.*

At each time step t do the following:

1. Compute the optimal schedule $\hat{S}$ for all packets not yet sent or expired (i.e., implicitly assuming no new packet will arrive).
2. Send the W packets which are sent at time t in $\hat{S}$ (we may assume, without loss of generality, that $\hat{S}$ is EDF).

FIG. 4.1. *The local-EDF algorithm.*

At each time step t do the following:

1. Compute the optimal EDF schedule $\hat{S}$ for all packets not yet sent or expired. Let S be the set of all packets scheduled in $\hat{S}$, and let S' and S'' be the sets of packets scheduled in $\hat{S}$ to be sent at times t and $t+1$, respectively.
2. Let $p \in S'$ be the packet of minimal value in S', and let $q \in S''$ be the packet of maximal value in S''.
3. If $v(p) < \beta \cdot v(q)$, then
   (a) **Push-out step:** Let $S' \leftarrow S' \cup \{q\} \setminus \{p\}$, and $S'' \leftarrow S'' \setminus \{q\}$. The packet q is said to have *pushed out* packet p.
   (b) Go to Step 2.
4. Otherwise (i.e., $v(p) \geq \beta \cdot v(q)$), terminate the algorithm and transmit S'.

FIG. 4.2. *The β-EDF algorithm.*

Proof. By definition, β-EDF sends packets with the same deadline in nondecreasing value order. □

LEMMA 4.10. *Consider the sets S' constructed at Step 1 at time t by the β-EDF algorithm. If some packet $p \in S_O(t)$ with deadline t is not included in S_G, then $(S' \cap A(t)) \subset S_O(t)$.*

Proof. Let $p' \in S' \cap A(t)$. Notice that $|S'| = W$ and $v(p') > v(p)$ because $p \notin S_G$. We argue that $p' \in S_O(t)$. First, note that $p' \in S_O$, since otherwise OPT could have been improved by swapping p and p'. Hence $p \in S_O(t) \cup S_O(t+1)$. To see that $p' \in S_O(t)$, suppose for contradiction that $p' \in S_O(t+1)$. Note that since p is not included in S_G, we must have that either S' contains no packet with deadline $t+1$, or $|S| = 2W$. In the former case, we get a contradiction to our assumption that $p' \in S'$ because the deadline of p' is $t+1$. In the latter case, we get that since $p' \in S' \cap S_O(t+1)$ and $|S| = 2W$, there exists a packet $p'' \in A(t) \cap S$ with deadline $t+1$ such that $p'' \notin S_O$. Moreover, $v(p'') > v(p)$ by construction of the β-EDF schedule. In this case, OPT can be improved by replacing p with p''. The lemma follows. □

For the remainder of the proof, we define a mapping $m : S_O \rightarrow S_G$ iteratively as follows.

1. If $p \in S_O(t) \cap (S_G(t-1) \cup S_G(t))$ for some time step t, then $m(p) = p$.
2. For each time step t, map any unmapped packet $p \in S_O(t)$ to $p' \in S_G(t)$ such that either
   (a) p' is unmapped, or
   (b) $p' \in S_O(t+1)$ and $|m^{-1}(p')| = 1$.

To prove the theorem, it suffices to show that (i) all packets in S_O are mapped, and that (ii) $v(m^{-1}(p)) \leq \phi \cdot v(p)$ for all $p \in S_G$.

LEMMA 4.11. *If at Step 2 a packet $p \in S_O(t)$ has to be mapped, then there always exists a packet $p' \in S_G$ eligible for either Step 2(a) or Step 2(b).*

Proof. Consider an unmapped packet $p \in S_O(t)$ that is processed in the course of Step 2. Since p has not been mapped at Step 1, either $p \in S_G(t+1)$ or $p \in S_O \setminus S_G$. Thus, since p is available to ϕ-EDF at time t and it is not transmitted at that time, we have that $|S_G(t)| = W$. The lemma follows by construction of the mapping. ∎

LEMMA 4.12. *If a packet $p \in S_O(t)$ is mapped to a packet p', then $v(p) \le v(p')/\phi$.*

Proof. Suppose first that p is mapped to a packet p' during Step 2(a). Then since p is available to ϕ-EDF at time t and it is not transmitted, the value of any packet that is scheduled at that time is at least $v(p) \cdot \phi$. So for the remainder of the proof, assume that p is mapped to p' during Step 2(b). In this case, since p has not been mapped at Step 1, either $p \in S_G(t+1)$ or $p \in S_O \setminus S_G$. By Lemma 4.7, $p \notin S_G(t+1)$. Also, p cannot have deadline $t+1$, since otherwise, $v(p) > v(p')$ and ϕ-EDF would have replaced p' by p. Hence $p \in (S_O \setminus S_G)$, and the deadline of p is t. It follows that it must be the case that either (1) p is pushed out at Step 3(a) of β-EDF, or (2) $p \notin S$, where S is the set computed at Step 1 of β-EDF. Let us analyze these cases.

(1) If p' performed a push-out, then we are done by Lemma 4.8. Otherwise, let q be the packet that pushed out p. According to Lemma 4.9 we have that $v(q) \le v(p')$, and consequently, $v(p) \le v(p')/\phi$.

(2) If $p \notin S$, then by Lemma 4.10 we have that $p' \notin S' \cap A(t)$ because $p' \notin S_O(t) \cap A(t)$ (recall that $p' \in S_O(t+1)$). Thus, if $p' \in S_G(t)$, then it should have pushed out some packet q such that $v(q) \le v(p)/\phi$. By construction of the optimal ϕ-EDF schedule we have that $v(q) \ge v(p)$. Hence, $v(p) \le v(p')/\phi$. ∎

Lemmas 4.11 and 4.12 conclude the proof of Theorem 4.6. ∎

Uniform delay buffers. We remark that for uniform-delay buffers with $\delta = 2$, better upper bounds can be obtained. For example, Corollary 5.5 (in conjunction with Theorem 3.4) says that a competitive ratio of 1.5 is achieved by the FIFO-greedy algorithm in this model. Moreover, careful case analysis shows that the β-EDF algorithm achieves a ratio of about 1.43 when $\beta = \frac{3+\sqrt{13}}{2} \approx 3.3$. We omit the details.

We now prove lower bounds for the case $\delta = 2$. First, we consider the case of arbitrary bandwidth.

THEOREM 4.13. *The competitive ratio of any on-line algorithm for a W-bandwidth 2-uniform bounded-delay model is at least* 10/9. *Moreover, the competitive ratio of any on-line algorithm for a W-bandwidth 2-variable bounded-delay model is at least* 1.17.

Proof. We first show the bound for the uniform model. Fix an on-line algorithm A and consider the following scenarios. Initially, the buffer is empty and $2W$ packets of value 1 arrive. At the next step W packets of value α arrive. Suppose that A drops a fraction $x \le 1$ of the 1-value packets. We consider two possible scenarios. In the first no more packets arrive. Then the competitive ratio is bounded from below by $\frac{\alpha+2}{\alpha+2-x}$ since there exists a feasible schedule of the whole sequence. In the second scenario $2W$ packets of value α arrive at the following time step. In this case the competitive ratio of A is bounded from below by $\frac{3\alpha+1}{(2+x)\alpha+2-x}$. Similar to the proof of Theorem 4.1, the "best" value of x is the one that equates the two ratios. In this case we get $x = \frac{(\alpha+2)(\alpha-1)}{\alpha^2+4\alpha-1}$. Substituting and maximizing for α we get $\alpha = 3$ and $x = 1/2$ to yield the ratio 10/9.

We now establish the bound for the variable model. Fix an on-line algorithm A and consider the following scenarios. Initially, the buffer is empty and W packets of value 1 and delay 1 arrive. At the same step, W packets of value α and delay 2 arrive. Suppose that A drops a fraction $x \le 1$ of the 1-value packets. We consider two possible scenarios. In the first no more packets arrive. Then the competitive

ratio is bounded from below by $\frac{\alpha+1}{\alpha+1-x}$ since there exists a feasible schedule of the whole sequence. In the second scenario, W packets of value α and delay 1 arrive at the next time step. In this case the competitive ratio of A is bounded from below by $\frac{2\alpha}{(1+x)\alpha+1-x}$. These are the same ratios considered in the proof of Theorem 4.1, and hence we get the same bound of $1+\frac{1}{(1+\sqrt{2})^2} \approx 1.17$. □

Slightly better results can be proved for bandwidth 1.

THEOREM 4.14. *The competitive ratio of any deterministic on-line algorithm for a* 2*-uniform and a* 2*-variable bounded-delay model with bandwidth* 1 *is at least* 1.25 *and* $\sqrt{2}$*, respectively.*

Proof. Consider the uniform model first. Fix an on-line algorithm A and consider the following scenario. At time 0, the buffer is empty and two packets of value 1 arrive, and at time 1, a packet of value $\alpha > 1$ arrives. There are two possible continuations. In one, no more packets arrive, and in the other, at time 2 two additional packets of value α arrive. Now, if A drops one of the low-value packets, then its competitive ratio is at least $\frac{\alpha+2}{\alpha+1}$ since there exists a feasible schedule of all three packets of the first continuation. Otherwise, at least one packet of value α is lost by A in the second continuation, and hence the competitive ratio of A is at least $\frac{3\alpha+1}{2\alpha+2}$. Setting $\frac{\alpha+2}{\alpha+1} = \frac{3\alpha+1}{2\alpha+2}$, we get that for $\alpha = 3$, the competitive ratio of A is at least 1.25.

The bad example for the variable delay model is even simpler. Let A be an on-line algorithm, and consider the following scenario. At time 0, the buffer is empty and a packet having value 1 and delay 1 arrives together with a packet of value $\alpha > 1$ and delay of 2. The two possible continuations are (i) no more arrivals and (ii) at time 1 an additional packet of value α with delay 1 (i.e., zero slack time) arrives. If A drops the low-value packet, then its competitive ratio for continuation (i) is $\frac{\alpha+1}{\alpha}$ since there exists a feasible schedule of both packets. If the low-value packet is scheduled, then for continuation (ii), A loses at least one high-value packet, showing that its competitive ratio is at least $\frac{2\alpha}{\alpha+1}$. Solving $\frac{\alpha+1}{\alpha} = \frac{2\alpha}{\alpha+1}$, we get that for $\alpha = 1+\sqrt{2}$, the competitive ratio of A is at least $\sqrt{2}$. □

5. The off-line case. In this section we show that the FIFO model has matroid structure in the off-line setting. As a result, optimal off-line solutions can be found in polynomial time. We also study the connection between the FIFO model and the bounded-delay model.

We first consider the FIFO model. Fix the input sequence for the remainder of this section. We also assume, without loss of generality, that all packets admitted to the buffer are later sent: since we are dealing with the off-line case, a packet that will be dropped can simply be rejected when it arrives.

Let $\mathcal{C}$ be the class of all work-conserving schedules, defined as follows. A schedule A is said to be *work conserving*, denoted by $A \in \mathcal{C}$, if

$$|S_A(t)| = \min(W,\ |Q_A(t) \cup A(t) \setminus D_A(t)|) \qquad \text{for all time steps } t,$$

where W is the link bandwidth. In words, a schedule is work-conserving if a packet may be delayed only when the full bandwidth is used by other packets. Note that work-conserving algorithms may still reject packets arbitrarily.

THEOREM 5.1. *For a FIFO schedule* A*, let* S_A *be the set of all packets served by* A*. Then* $\mathfrak{I}_{FIFO} \stackrel{\text{def}}{=} \{S_A : A \in \mathcal{C}\}$ *is a matroid.*

Proof. There are three properties to verify. The first two are trivial: $\emptyset \in \mathfrak{I}$, and for $S_A \subset S_B$ with $S_B \in \mathfrak{I}$, we clearly have that $S_A \in \mathfrak{I}$ by dropping the packets in $B \setminus A$. It remains to verify the following property: If $|S_A| > |S_B|$, then there exists $p \in S_A \setminus S_B$

such that $S_B \cup \{p\} \in \mathfrak{I}$. This can be seen as follows. Let t_0 be the first time in which $|S_A(t_0)| > |S_B(t_0)|$: t_0 exists by the assumption that $|S_A| > |S_B|$. Let $t_1 \le t_0$ be the last step before t_0 where $|D_A(t_1)| < |D_B(t_1)|$: t_1 exists by the assumption that B is work conserving. Then there exists a packet $p \in D_B(t_1) \setminus D_A(t_1)$. Moreover, since by our choice, for any $t \in [t_1, t_0]$ we have that $|S_B(t)| \ge |S_A(t)$ and $|D_B(t)| \le |D_A(t)|$, and since A is work conserving by assumption, we also have that $|Q_B(t)| < |Q_A(t)|$ for all $t \in [t_1, t_0]$. Hence the packet p can be added to S_B while keeping the schedule feasible. □

A similar result for the bounded delay case is well known [10, Theorem 17.12]. We state it here for completeness.

THEOREM 5.2. *For a bounded delay schedule A, let S_A be the set of all packets served by A. Then $\mathfrak{I}_{BD} \stackrel{\text{def}}{=} \{S_A : A \in \mathcal{C}\}$ is a matroid.*

We remark that for the FIFO model (and hence for the uniform bounded-delay model as well; see Theorem 5.4 below), an optimal solution can be found in $O(n \log B)$ time, and in $O(n^2)$ time for the variable bounded delay model, where n is the number of packets in the input sequence and B is the buffer size.

COROLLARY 5.3. *An optimal schedule for the FIFO and bounded delay models can be found in polynomial time.*

The following theorem shows a transformation from the FIFO model to the uniform bounded-delay model.

THEOREM 5.4. *For any input sequence, the optimal value served by a FIFO schedule with buffer size B is equal to the optimal value served by a uniform bounded-delay schedule with $\delta = B + 1$.*

Proof. Let OPT_F be the optimal value served by a FIFO schedule with buffer size B, and let OPT_D be the optimal value served by a uniform bounded-delay schedule with $\delta = B + 1$. First, note that any work-conserving schedule in the FIFO model is also a schedule in the bounded delay model, since no packet in the FIFO model is served more than B time units after its arrival, and hence $OPT_F \le OPT_D$. For the other direction, consider any schedule in the uniform bounded-delay model. Since in this model, a set of packets can be served if and only if the EDF schedule of this set is feasible, we may assume without loss of generality that the schedule is EDF. Also note that the number of packets in the bounded delay buffer is never more than the maximal delay bound (recall that only packets that are eventually transmitted enter the buffer). The result now follows from the fact that an EDF schedule in the uniform bounded delay model is exactly the FIFO order, and hence $OPT_D \le OPT_F$. □

A nice feature of the FIFO to bounded-delay transformation in the proof of Theorem 5.4 is that it does not require off-line information. We therefore have the following corollary.

COROLLARY 5.5. *Let $C_F(B)$ be the best competitive factor of on-line FIFO algorithms with buffer size B, and let $C_D(\delta)$ be the best competitive factor of on-line uniform bounded-delay algorithms with maximal delay δ. Then $C_D(B+1) \le C_F(B)$.*

We remark that the converse cannot be proved by our transformation, since it requires knowledge of the future.

6. Conclusion. In this work we studied competitive overflow management. We sharpened the results of [20] for the FIFO model, and initiated a study in the bounded-delay model. In particular, we have proved the following facts:

- The greedy algorithm is 2-competitive for FIFO buffers.
- Among all the greedy algorithms, head-drop is the best.
- No on-line algorithm can be optimal for the bounded-delay case.

- The greedy algorithm is $(1+\frac{1}{\alpha})$-competitive in the bounded delay model when the set of possible values is 1 and $\alpha > 1$.

Many important questions remain open:

- In the FIFO model, can one substantially improve on the 2-competitiveness of the greedy algorithm in the general case? The best known result [14] is for the case of two packet values 1 and $\alpha > 4$, with competitive ratio of $\frac{\sqrt{\alpha}+1}{\sqrt{\alpha}}$. Combining this with our results for the greedy algorithm, one can get a competitive ratio better than 2 for this case only. There is no real improvement for the general case of packet values.
- In the bounded delay model, we know very little in the general case. Even for the uniform bounded delay case, we know how to improve the greedy algorithm only for the special case of $\delta = 2$.

REFERENCES

[1] M. Adler, A. L. Rosenberg, R. K. Sitaraman, and W. Unger, *Scheduling time-constrained communication in linear networks*, in Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures, 1998, pp. 269–278.

[2] W. Aiello, Y. Mansour, S. Rajagopolan, and A. Rosen, *Competitive queue policies for differentiated services*, in Proc. IEEE INFOCOM, 2000, pp. 431–440.

[3] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, *On-line routing of virtual circuits with applications to load balancing and machine scheduling*, J. ACM, 44 (1997), pp. 486–504.

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

[5] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, *Approximating the throughput of multiple machines in real-time scheduling*, SIAM J. Comput., 31 (2001), pp. 331–352.

[6] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, *On the competitiveness of online real-time task scheduling*, in Proceedings of the 32nd IEEE Symposium on Real-Time Systems, 1992, pp. 125–144.

[7] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, *Online scheduling in the presence of overload*, in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 101–110.

[8] D. Black, S. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Services*, Internet draft, RFC 2475, Internet Engineering Task Force, 1998.

[9] D. Clark and J. Wroclawski, *An Approach to Service Allocation in the Internet*, unpublished, 1997.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[11] S. Floyd and V. Jacobson, *Random early detection gateways for congestion avoidance*, IEEE/ACM Trans. on Networking, 1 (1993), pp. 397–413.

[12] S. Goldman, J. Parwatikar, and S. Suri, *On-line scheduling with hard deadlines*, J. Algorithms, 34 (2000), pp. 370–389.

[13] V. Jacobson, K. Nicholas, and K. Poduri, *An Expedited Forwarding PHB*, Internet draft, RFC 2598, Internet Engineering Task Force, 1999.

[14] A. Kesselman and Y. Mansour, *Loss-bounded analysis for differentiated services*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 591–600.

[15] G. Koren and D. Shasha, *D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems*, SIAM J. Comput., 24 (1995), pp. 318–339.

[16] M. A. Labrador and S. Banerjee, *Packet dropping policies for ATM and IP networks*, IEEE Communications Surveys, 2 (1999).

[17] T. V. Lakshman, A. Neidhardt, and T. Ott, *The drop from front strategy in TCP and in TCP over ATM*, in Proc. IEEE INFOCOM, 1996, pp. 1242–1250.

[18] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, *On the self-similar nature of ethernet traffic (extended version)*, IEEE/ACM Trans. on Networking, 2 (1994), pp. 1–15.

[19] R. Lipton and A. Tomkins, *Online interval scheduling*, in Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1994, pp. 302–311.

[20] Y. Mansour, B. Patt-Shamir, and O. Lapid, *Optimal smoothing schedules for real-time streams*, Distributed Computing, 17 (2004), pp. 77–89.

[21] M. May, J.-C. Bolot, A. Jean-Marie, and C. Diot, *Simple performance models of differentiated services for the Internet*, in Proc. IEEE INFOCOM, 1999, pp. 1385–1394.

[22] C. Phillips, R. Uma, and J. Wein, *Off-line admission control for general scheduling problems*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2000, pp. 879–888.

[23] The ATM Forum Technical Committee, *Traffic Management Specification Version* 4.0, ftp://ftp.atmforum.com/pub/approved-specs/af-tm-0056.000.pdf (April 1996).

[24] A. Veres and M. Boda, *The chaotic nature of TCP congestion control*, in Proc. IEEE INFOCOM, 2000, pp. 1715–1723.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 584–589

ON THE FRAME–STEWART CONJECTURE ABOUT THE TOWERS OF HANOI*

XIAO CHEN† AND JIAN SHEN‡

Abstract. The multipeg Towers of Hanoi problem consists of k pegs mounted on a board together with n disks of different sizes. Initially these disks are placed on one peg in the order of their size, with the largest at the bottom. The rules of the problem allow disks to be moved one at a time from one peg to another as long as a disk is never placed on top of a smaller disk. The goal of the problem is to transfer all the disks to another peg with the minimum number of moves, denoted by $H(n,k)$. An easy recursive argument shows that $H(n,3) = 2^n - 1$. However, the problem of computing the exact value of $H(n,k)$ for $k \geq 4$ has been open since 1939, and in particular, the special case of $H(n,4)$ has been open since 1907.

In 1941, Frame and Stewart each gave an algorithm to solve the Towers of Hanoi problem based on an unproved assumption. The Frame–Stewart number, denoted by $FS(n,k)$, is the number of moves needed to solve the Towers of Hanoi problem using the "presumed optimal" Frame–Stewart algorithm. Since then, proving the Frame–Stewart conjecture $FS(n,k) = H(n,k)$ has become a notorious open problem.

In this paper, we prove that $FS(n,k)$ and $H(n,k)$ both have the same order of magnitude of $2^{(1\pm o(1))(n(k-2)!)^{1/(k-2)}}$. This provides the strongest evidence so far to support the Frame–Stewart conjecture.

Key words. Towers of Hanoi problem, Frame–Stewart conjecture, optimal algorithm, Frame–Stewart algorithm

AMS subject classifications. 05A10, 05A16, 68R05

DOI. 10.1137/S0097539703431019

1. Introduction.

The Towers of Hanoi problem, introduced by Édouard Lucas in 1883, consists of three pegs and a set of n disks of different diameters that can be stacked on the pegs. The towers are formed initially by stacking the disks onto one peg in the order of their size, with the largest at the bottom. The rules of the problem allow disks to be moved one at a time from one peg to another as long as a disk is never placed on the top of a smaller disk. The goal of the problem is to transfer all the disks to another peg with the minimum number of moves. An easy argument using a recursive relation shows that $2^n - 1$ moves are necessary and sufficient to carry out this task.

The Towers of Hanoi problem was extended to four pegs by Dudeney [2] in 1907 and to any arbitrary $k \geq 3$ pegs by Stewart [7] in 1939. In 1941, Frame [3] and Stewart [8] independently proposed an algorithm to the Towers of Hanoi problem with $k \geq 4$ pegs:

1. Recursively transport a stack of $n - i$ smallest disks from the first peg to a temporary peg, using all k pegs;
2. Transport the remaining stack of i largest disks from the first peg to the final peg, using $k - 1$ pegs and ignoring the peg occupied by the smaller disks;

*Received by the editors July 10, 2003; accepted for publication (in revised form) January 16, 2004; published electronically March 23, 2004.
http://www.siam.org/journals/sicomp/33-3/43101.html
†Department of Computer Science, Texas State University-San Marcos, San Marcos, TX 78666 (xc10@txstate.edu).
‡Department of Mathematics, Texas State University-San Marcos, San Marcos, TX 78666 (js48@txstate.edu).

3. Recursively transport the smallest $n-i$ disks from the temporary peg to the final peg, using all k pegs.

(Indeed, Frame's algorithm is slightly different from the above proposed by Stewart. But both algorithms are essentially equivalent [4].) The Frame–Stewart number, denoted by $FS(n,k)$, is the minimum number of moves needed to solve the Towers of Hanoi problem using the above Frame–Stewart algorithm. Thus $FS(n,k)$ has the following recursive formula:

$$FS(n,k) = \begin{cases} 2^n - 1 & \text{if } k = 3, \\ \min_{1 \le i < n} \{2FS(n-i,k) + FS(i,k-1)\} & \text{if } k \ge 4. \end{cases}$$

The Frame–Stewart number $FS(n,k)$ is called the "presumed optimal" solution since no justification has ever been made that an optimal algorithm must be of this form. Let $H(n,k)$ be the minimum number of moves needed to solve the Towers of Hanoi problem. The Frame–Stewart conjecture $FS(n,k) = H(n,k)$ is still open now. (As pointed out by Klavžar, Milutinović, and Petr [4], the claimed proof of the conjecture by Majumdar [6] is indeed incorrect.) Donald Knuth commented on the conjecture, saying "I doubt if anyone will ever resolve the conjecture; it is truly difficult" (see [5]).

In the attempt to prove the Frame–Stewart conjecture, Bode and Hinz [1] verified the conjecture for four pegs with up to 17 disks. Recently, Szegedy [9] proved that

$$FS(n,k) \ge 2^{(1 \pm o(1)) c_k n^{1/(k-2)}},$$

where $c_k = \frac{1}{2}\left(\frac{12}{k(k-1)}\right)^{1/(k-2)}$.

For convenience, let $\log x$ denote the logarithmic function with base 2. In this paper, we prove that for $n \ge 1$ and $k \ge 3$,

$$\log FS(n,k) = \log H(n,k) + \Theta(k + \log n) = (n(k-2)!)^{1/(k-2)} + \Theta(k + \log n).$$

In other words, for each fixed $k \ge 3$ and for $n \gg k$,

$$FS(n,k) = 2^{(1 \pm o(1))(n(k-2)!)^{1/(k-2)}} = H(n,k).$$

This provides the strongest evidence so far to support the Frame–Stewart conjecture.

2. Lower bound on the optimal number of moves. In this section, we derive a lower bound on the optimal number $H(n,k)$ of moves for the Towers of Hanoi problem. We adopt the remarkable strategy introduced by Szegedy [9] who considered the following generalized problem: What is the minimum number of moves to move each disk at least once among all possible initial setups of disks? The advantage of this strategy is that one can use induction in the proofs.

An arrangement of k pegs and n disks is called a configuration if it obeys the "smaller disk on the top of larger disk" rule. For a configuration C, let $g(C)$ be the minimum number of moves required to have every disk moved at least once, where all moves are taken according to the rules of the Towers of Hanoi. Szegedy [9] defined

$$g(n,k) = \min_C g(C),$$

where C runs through all possible configurations of n disks and k pegs. The function $g(n,k)$ is well defined since $g(C)$ is finite for some configuration C [9, Remark 1]. By the definition of $g(n,k)$, we have $H(n,k) \ge g(n,k)$, and thus a lower bound on

$H(n,k)$ can be derived from Theorem 2. We begin with a study of the monotone properties of the function $g(n,k)$.

LEMMA 1. *Suppose $n \geq 1$ and $k \geq 3$. Then the function $g(n,k)$ is decreasing with respect to the variable k.*

THEOREM 1. *Suppose $n \geq 2$ and $k \geq 4$. Then there exists some m with $1 \leq m \leq n-1$ such that*

$$g(n,k) \geq \begin{cases} 2\max\{g(n-m,k), g(m,k-1)\} & \text{if } g(n-m,k) = g(m,k-1), \\ 2\max\{g(n-m,k), g(m,k-1)\} - 1 & \text{if } g(n-m,k) \neq g(m,k-1). \end{cases}$$

Proof. Let C be an extremal configuration of n disks and k pegs with $g(C) = g(n,k)$. Let $S = \left(s_1, s_2, \ldots, s_{g(C)}\right)$ be a sequence of $g(C)$ moves that move every disk of C at least once, where all moves are taken according to the rules of the Towers of Hanoi. Let $S = S_1 \cup S_2$ with $|S_1| = \lfloor g(C)/2 \rfloor$ and $|S_2| = \lceil g(C)/2 \rceil$; that is, S_1 and S_2 are the first half and the second half of the sequence of moves of S, respectively. For $i = 1, 2$, let

$$D_i = \{j : \text{disk } j \text{ is moved at least once by } S_i\}.$$

Then $|D_1 \cup D_2| = n$.

Claim. $D_1 - D_2 \neq \emptyset$ and $D_2 - D_1 \neq \emptyset$.

Proof of the claim. Let $s_1(C)$ be the configuration obtained by applying the first move s_1 of S to C. Suppose the move s_1 moves disk i. We observe that disk i cannot be moved more than once by S; otherwise, each disk in the configuration $s_1(C)$ can be moved at least once by the following $g(C) - 1$ $(= g(n,k) - 1)$ moves $s_2, \ldots, s_{g(C)}$, contradicting the definition of $g(n,k)$. Since disk i is moved only by $s_1 \in S_1$, we have $i \in D_1 - D_2 \neq \emptyset$. Similarly, the disk moved by the last move $s_{g(C)}$ of S cannot be moved more than once by S either. Thus $D_2 - D_1 \neq \emptyset$.

Let $D_1 - D_2 = \{r_1, \ldots, r_l\}$ and $D_2 - D_1 = \{t_1, \ldots, t_m\}$, where $1 \leq l, m \leq n-1$. We may label the disks in such a way that disk i has larger size than disk j if and only if $i > j$. Let r_1 be the smallest number in $(D_1 - D_2) \cup (D_2 - D_1)$. Since $|D_1| = |D_1 \cup D_2| - |D_2 - D_1| = n - m$, by the definition of D_1, we know that S_1 moves $n - m$ different pegs. Suppose the moves of S_1 take place in t pegs, where $t \leq k$. Then, by Lemma 1,

$$|S_1| \geq g(n-m,k). \tag{1}$$

Since $r_1 \in D_1 - D_2$, disk r_1 is not moved by S_2. Since $r_1 < t_i$ for all $1 \leq i \leq m$, all disks t_i $(1 \leq i \leq m)$ have larger sizes than disk r_1, which is idle during the whole movement of S_2. By the "smaller disk on the top of larger disk" rule, the peg occupied by disk r_1 is completely useless when each disk t_i $(1 \leq i \leq m)$ is moved by S_2. Thus the m disks $t_1, \ldots, t_m$ are moved by S_2 using at most $k-1$ pegs. (S_2 might also move disks other than disks $t_1, \ldots, t_m$. But those moves and disks can be ignored since they do not affect the moves involving disks $t_1, \ldots, t_m$. So one can focus on only the subsequence of S_2 that moves disks $t_1, \ldots, t_m$.) By Lemma 1,

$$|S_2| \geq g(m,k-1). \tag{2}$$

Note that $g(n,k) \geq 2\max\{|S_1|, |S_2|\} - 1$. If $g(n-m,k) \neq g(m,k-1)$, then Theorem 1 follows from (1) and (2). If $g(n-m,k) = g(m,k-1)$, then $g(n,k) \geq 2|S_1| \geq 2g(n-m,k) = 2\max\{g(n-m,k), g(m,k-1)\}$. ∎

LEMMA 2. *Suppose $n \geq 1$ and $k \geq 3$. Then the function $g(n,k)$ is strictly increasing with respect to the variable n.*

Proof. Let C be an extremal configuration of n disks and k pegs with $g(C) = g(n,k)$. Let $S = \left(s_1, s_2, \ldots, s_{g(C)}\right)$ be a sequence of $g(C)$ moves that move every disk of C at least once, where all moves are taken according to the rules of the Towers of Hanoi. Let $s_1(C)$ be the configuration obtained by applying the first move s_1 of S to C. Suppose the move s_1 moves disk i. In the proof of Theorem 1, it is shown that disk i cannot be moved more than once by S. Then every disk except disk i in the configuration of $s_1(C)$ is moved at least once by $S - \{s_1\}$, which consists of a sequence of $g(n,k) - 1$ moves. Let C' be the configuration obtained by removing the disk i from the configuration $s_1(C)$. Then C' has $n-1$ disks and k pegs, and $g(n-1,k) \leq g(C') = g(C) - 1 = g(n,k) - 1$. □

COROLLARY 1. *Suppose $n \geq 2$ and $k \geq 4$. Then for every m with $1 \leq m \leq n-1$,*

$$g(n,k) \geq 2\min\{g(n-m,k), g(m,k-1)\}.$$

Proof. By Lemma 2, $g(n-m,k)$ is a strictly decreasing function of m, and $g(m,k-1)$ is a strictly increasing function of m. Corollary 1 obviously follows from Theorem 1. □

As usual, the function $\binom{x}{t}$ can be extended to real x for each integer t as follows:

$$\binom{x}{t} = \begin{cases} 0 & \text{if } t < 0, \\ 1 & \text{if } t = 0, \\ x(x-1)\cdots(x-t+1)/t! & \text{if } t > 0. \end{cases}$$

In particular, $\binom{0}{0} = 1$ by the above definition. The identity $\binom{x}{t} = \binom{x-1}{t} + \binom{x-1}{t-1}$ will be used repeatedly in the proofs.

THEOREM 2. *Suppose $k \geq 3$. Then for every integer $s \geq 2$,*

$$g\left(\binom{s}{k-2} + \binom{s+k-7}{k-5}, k\right) \geq 2^{s-2}.$$

Proof. We use double-induction on k and s. First, we use induction on k. If $k = 3$, by [9, Remark 2],

$$g\left(\binom{s}{k-2} + \binom{s+k-7}{k-5}, k\right) = g(s,3) \geq 2^{s-2} + 1$$

for all $s \geq 2$. Now suppose $k \geq 4$ and suppose the theorem is true for $k-1$; that is, for every integer $s \geq 2$,

$$g\left(\binom{s}{k-3} + \binom{s+k-8}{k-6}, k-1\right) \geq 2^{s-2}.$$

Equivalently, by using $s-1$ to replace s in the above, we have

$$g\left(\binom{s-1}{k-3} + \binom{s+k-9}{k-6}, k-1\right) \geq 2^{s-3} \tag{3}$$

for all $s \geq 3$.

Second, we use induction on s. If $s = 2$, then $\binom{s}{k-2} + \binom{s+k-7}{k-5} = \binom{2}{k-2} + \binom{k-5}{k-5} = 1$ since $k \geq 4$. Thus

$$g\left(\binom{s}{k-2} + \binom{s+k-7}{k-5}, k\right) = g(1,k) = 1 = 2^{s-2};$$

that is, Theorem 2 is true for $s = 2$ and $k \geq 4$. Now suppose $s \geq 3$ and suppose the theorem is true for $s - 1$; that is, for every integer $k \geq 4$,

$$g\left(\binom{s-1}{k-2} + \binom{s+k-8}{k-5}, k\right) \geq 2^{s-3}. \tag{4}$$

Let $n = \binom{s}{k-2} + \binom{s+k-7}{k-5}$ and $m = \binom{s-1}{k-3} + \binom{s+k-8}{k-6}$. Then by Corollary 1 together with (3) and (4),

$$\begin{aligned} g(n,k) &\geq 2\min\{g(n-m,k), g(m,k-1)\} \\ &= 2\min\left\{g\left(\tbinom{s-1}{k-2} + \tbinom{s+k-8}{k-5}, k\right), g\left(\tbinom{s-1}{k-3} + \tbinom{s+k-8}{k-6}, k-1\right)\right\} \\ &\geq 2\min\left\{g\left(\tbinom{s-1}{k-2} + \tbinom{s+k-8}{k-5}, k\right), g\left(\tbinom{s-1}{k-3} + \tbinom{s+k-9}{k-6}, k-1\right)\right\} \\ &\geq 2^{s-2}; \end{aligned}$$

that is, Theorem 2 is true for s and k. The proof is complete by the principle of double-induction. □

3. Proof of main result. By the definition of $g(n,k)$ and $H(n,k)$, we have $g(n,k) \leq H(n,k) \leq FS(n,k)$. Thus, in order to obtain the order of magnitude of $H(n,k)$, one needs to have a lower bound on $g(n,k)$ and an upper bound on $FS(n,k)$ with the same order of magnitude.

LEMMA 3. *Suppose $n \geq 1$ and $k \geq 3$. Then*

$$\log FS(n,k) < (n(k-2)!)^{1/(k-2)} + \log n.$$

Proof. For any fixed k and n, there is a unique s such that $\binom{k+s-3}{k-2} < n \leq \binom{k+s-2}{k-2}$. The exact expression on $FS(n,k)$ in the first line below can be found in many papers (for example, [3]).

$$\begin{aligned} FS(n,k) &= 2^s\left(n - \tbinom{k+s-3}{k-2}\right) + \sum_{t=0}^{s-1} 2^t \tbinom{k+t-3}{k-3} \\ &\leq 2^s\left(n - \tbinom{k+s-3}{k-2}\right) + \tbinom{k+s-4}{k-3}\sum_{t=0}^{s-1} 2^t \\ &< 2^s\left(n - \tbinom{k+s-3}{k-2}\right) + \tbinom{k+s-4}{k-3}2^s \\ &= 2^s\left(n - \tbinom{k+s-3}{k-2} + \tbinom{k+s-4}{k-3}\right) \\ &= 2^s\left(n - \tbinom{k+s-4}{k-2}\right) < n2^s. \end{aligned}$$

Thus $\log FS(n,k) \leq s + \log n$. To estimate s, we have $n > \binom{k+s-3}{k-2} > s^{k-2}/(k-2)!$, which implies $s < (n(k-2)!)^{1/(k-2)}$. □

LEMMA 4. *Suppose $n \geq 1$ and $k \geq 3$. Then*

$$\log g(n,k) > (n(k-2)!)^{1/(k-2)} - k + 1.$$

Proof. Lemma 4 holds for $n = 1$ since

$$\log g(1,k) = \log 1 = 0 > ((k-2)!)^{1/(k-2)} - k + 1.$$

If $k = 3$, by [9, Remark 2], we have $g(n, 3) \geq 2^{n-2} + 1$ for all $n \geq 2$. Thus

$$\log g(n, 3) > n - 2 = (n(k-2)!)^{1/(k-2)} - k + 1.$$

Now suppose $n \geq 2$ and $k \geq 4$. Then there is a unique s such that $\binom{s}{k-2} + \binom{s+k-7}{k-5} < n \leq \binom{s+1}{k-2} + \binom{s+k-6}{k-5}$. Also it is easy to verify that $s \geq 2$. To estimate s, we have

$$\begin{aligned} n &\leq \tbinom{s+1}{k-2} + \tbinom{s+k-6}{k-5} \leq \begin{cases} \tbinom{s+1}{k-2} & \text{if } k = 4, \\ \tbinom{s+k-4}{k-2} + \tbinom{s+k-4}{k-3} & \text{if } k \geq 5 \end{cases} \\ &= \tbinom{s+k-3}{k-2} < (s+k-3)^{k-2}/(k-2)!, \end{aligned}$$

from which $s > (n(k-2)!)^{1/(k-2)} - k + 3$. By Lemma 2 and Theorem 2,

$$\log g(n, k) > \log g\left(\binom{s}{k-2} + \binom{s+k-7}{k-5}, k\right) \geq s-2 > (n(k-2)!)^{1/(k-2)} - k + 1. \qquad \square$$

Finally, we have the main theorem (Theorem 3) showing that $FS(n, k)$ and $H(n, k)$ both have the order of magnitude of $2^{(1\pm o(1))(n(k-2)!)^{1/(k-2)}}$.

THEOREM 3. *Suppose $n \geq 1$ and $k \geq 3$. Then*

$$\log FS(n, k) = \log H(n, k) + \Theta(k + \log n) = (n(k-2)!)^{1/(k-2)} + \Theta(k + \log n).$$

In other words, for each fixed k and for $n \gg k$,

$$FS(n, k) = 2^{(1\pm o(1))(n(k-2)!)^{1/(k-2)}} = H(n, k).$$

Proof. By the definition of $g(n, k)$ and $H(n, k)$, we have $g(n, k) \leq H(n, k) \leq FS(n, k)$. By Lemmas 3 and 4,

$$(n(k-2)!)^{1/(k-2)} - k + 1 < \log H(n, k) \leq \log FS(n, k) < (n(k-2)!)^{1/(k-2)} + \log n,$$

from which Theorem 3 follows. $\square$

Acknowledgment. Jian Shen wants to thank Professor Mario Szegedy for providing reference [9].

REFERENCES

[1] J. P. BODE AND A. M. HINZ, *Results and open problems on the Towers of Hanoi*, Congr. Numer., 139 (1999), pp. 113–122.
[2] H. DUDENEY, *The Canterbury Puzzles*, Thomas Nelson & Sons, London, 1907.
[3] J. S. FRAME, *Solution to advanced problem* 3918, Amer. Math. Monthly, 48 (1941), pp. 216–217.
[4] S. KLAVŽAR, U. MILUTINOVIĆ, AND C. PETR, *On the Frame-Stewart algorithm for the multi-peg Towers of Hanoi problem*, Discrete Appl. Math., 120 (2002), pp. 141–157.
[5] W. F. LUNNON, *The Reve's puzzle*, Comput. J., 29 (1986), p. 478.
[6] A. A. K. MAJUMDAR, *The generalized p-peg Towers of Hanoi problem*, Optimization, 32 (1995), pp. 175–183.
[7] B. M. STEWART, *Advanced problem* 3918, Amer. Math. Monthly, 46 (1939), p. 363.
[8] B. M. STEWART, *Solution to advanced problem* 3918, Amer. Math. Monthly, 48 (1941), pp. 217–219.
[9] M. SZEGEDY, *In how many steps the k peg version of the Towers of Hanoi game can be solved?*, in STACS 99 (Trier), Lecture Notes in Comput. Sci. 1563, Springer-Verlag, Berlin, 1999, pp. 356–361.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 590–607

INCOMPLETE DIRECTED PERFECT PHYLOGENY*

ITSIK PE'ER†, TAL PUPKO‡, RON SHAMIR§, AND RODED SHARAN¶

Abstract. Perfect phylogeny is one of the fundamental models for studying evolution. We investigate the following variant of the model: The input is a species-characters matrix. The characters are binary and directed; i.e., a species can only gain characters. The difference from standard perfect phylogeny is that for some species the states of some characters are unknown. The question is whether one can complete the missing states in a way that admits a perfect phylogeny. The problem arises in classical phylogenetic studies, when some states are missing or undetermined. Quite recently, studies that infer phylogenies using inserted repeat elements in DNA gave rise to the same problem. Extant solutions for it take time $O(n^2m)$ for n species and m characters. We provide a graph theoretic formulation of the problem as a graph sandwich problem, and give near-optimal $\tilde{O}(nm)$-time algorithms for the problem. We also study the problem of finding a single, general solution tree, from which any other solution can be obtained by node splitting. We provide an algorithm to construct such a tree, or determine that none exists.

Key words. perfect phylogeny, incomplete data, graph sandwich, evolution

AMS subject classifications. 05C85, 05C50, 92D15

DOI. 10.1137/S0097539702406510

1. Introduction.

When studying evolution, the divergence patterns leading from a single ancestor species to its contemporary descendants are usually modeled by a tree structure, called *phylogenetic tree*, or *phylogeny*. Extant species correspond to the tree leaves, while their common progenitor corresponds to the root. Internal nodes correspond to hypothetical ancestral species, which putatively split up and evolved into distinct species. Tree branches model changes through time of the hypothetical ancestor species. The common case is that one has information regarding the leaves, from which the phylogenetic tree is to be inferred. This task, called *phylogenetic reconstruction* (cf. [8]), was one of the first algorithmic challenges posed by biology, and the computational community has been dealing with problems of this flavor for over three decades (see, e.g., [13]).

The character-based approach to tree reconstruction describes extant species by their attributes or *characters*. Each character takes on one of several possible *states*. The input is represented by a matrix $\mathcal{A}$, where a_{ij} is the state of character j in

*Received by the editors April 29, 2002; accepted for publication (in revised form) December 19, 2003; published electronically March 23, 2004. Portions of this paper appeared as *Incomplete directed perfect phylogeny*, in Proceedings of the Eleventh Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 1848, Springer-Verlag, Berlin, 2000, pp. 143–153. Other parts appeared as *On the generality of phylogenies from incomplete directed characters*, in Proceedings of the Eighth Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 2368, Springer-Verlag, New York, 2002, pp. 358–367.

http://www.siam.org/journals/sicomp/33-3/40651.html

†Medical and Population Genetics Group, Broad Institute, 9 Cambridge Center, Cambridge, MA 02142 (peer@broad.mit.edu). This author's research was supported by a Clore foundation scholarship.

‡Department of Cell Research and Immunology, George S. Wise Faculty of Life Sciences, Tel Aviv University, Tel Aviv 69978, Israel (talp@post.tau.ac.il).

§School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (rshamir@tau.ac.il). This author's research was supported in part by a grant from the Israel Science Foundation (grant 309/02).

¶Corresponding author. International Computer Science Institute, 1947 Center St., Berkeley, CA 94704 (roded@icsi.berkeley.edu). This author's research was supported by a Fulbright grant and by an Eshkol fellowship from the Ministry of Science, Israel.

species i, and the ith row is the *character vector* of species i. The output sought is a hypothesis regarding evolution, i.e., a phylogenetic tree along with the suggested character vectors of the internal nodes. This output must satisfy properties specified by the problem variant.

One important class of phylogenetic reconstruction problems concerns finding a *perfect phylogeny*. The property required from such a phylogeny is that for each possible character state, the set of all nodes that have that state induces a connected subtree. The general perfect phylogeny problem is NP-hard [5, 21]. When considering the number of possible states per character as a parameter, the problem is fixed parameter tractable [2, 16]. For *binary characters*, having only two possible states, perfect phylogeny is linear-time solvable [12].

When no perfect phylogeny is possible, one option is to seek a largest subset of characters which admits a perfect phylogeny. Characters in such a subset are said to be *compatible*. Compatibility problems have been studied extensively (see, e.g., [18]).

Another common optimization approach to phylogenetic reconstruction is the *parsimony* criterion. It calls for a solution with the fewest state changes altogether, counting a change whenever the state of a character changes between a species and its ancestor species. This problem is known to be NP-hard [9]. A variant introduced by Camin and Sokal [6] assumes that characters are binary and *directed*, namely, only $0 \rightarrow 1$ changes may occur on any path from the root to a leaf. Denoting by 1 and 0 the presence and absence, respectively, of the character, this means that characters can only be gained throughout evolution. Another related binary variant is Dollo parsimony [7, 20], which assumes that a $0 \rightarrow 1$ change may happen only once; i.e., a character can be gained once, but it can be lost several times. Both of these variants are polynomially solvable (cf. [8]).

In this paper, we discuss a variant of binary perfect phylogeny which combines assumptions of both Camin–Sokal parsimony and Dollo parsimony. The setup is as follows: The characters are binary, directed, and can be gained only once. As in perfect phylogeny, the input is a matrix of character vectors, with the difference that some character states are missing. The question is whether one can complete the missing states in a way admitting a perfect phylogeny. We call this problem *Incomplete Directed Perfect phylogeny (IDP)*.

The problem of handling incomplete phylogenetic data arises whenever some of the data are missing. It is also encountered in the context of morphological characters, where for some species it may be impossible to reliably assign a state to a character. The problem of determining whether a set of incomplete *undirected* characters is compatible was shown to be NP-complete, even in the case of binary characters [21]. Indeed, the popular PAUP software package [22] provides an exponential solution to the problem by exhaustively searching the space of missing states.

Quite recently, a novel kind of genomic data has given rise to the same problem: Nikaido, Rooney, and Okado [19] use inserted repetitive genomic elements, particularly Short Interspersed Nuclear Elements (SINEs), as a source of evolutionary information. SINEs are short DNA sequences that were copied and randomly reinserted into various genomic loci during evolution. The distinct insertion loci are identifiable by the flanking sequences on both sides of the insertion site (see Figure 1). These insertions are assumed to be unique events in evolution, because the odds of having separate insertion events at the very same locus are negligible. Furthermore, a SINE insertion is assumed to be irreversible; i.e., once a SINE sequence has been inserted somewhere along the genome, it is practically impossible for the exact, complete SINE to leave

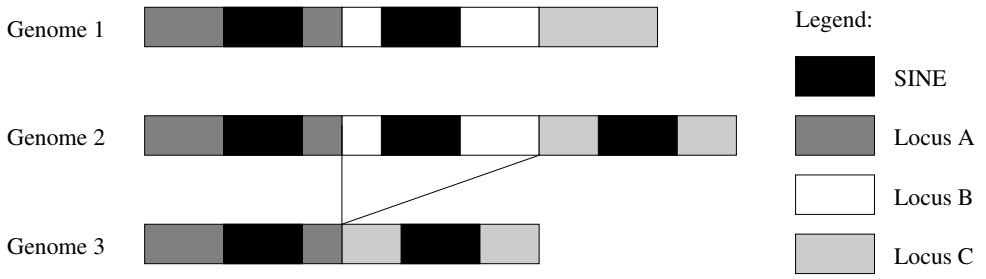


FIG. 1. *SINEs (black boxes) repeat in different loci (different shades of grey) across distinct genomes. A SINE insertion transformed Genome* 1 *into Genome* 2*. A deletion of a locus transformed Genome* 2 *into Genome* 3*. Given Genomes* 1 *and* 3*, we can identify that the SINE on locus C is not present in Genome* 1*, by its flanking sequence. However, locus B is missing in Genome* 3*.*

that specific locus. However, the inserted segment along with its flanking sequences may be lost when a large genomic region, which includes them, is deleted. In that case we do not know whether a SINE insertion had occurred in the missing site prior to its deletion. One can model such data by assigning to each locus a character, whose state is "1" if the SINE occurred in that locus, "0" if the locus is present but does not contain the SINE, and "?" if the locus is missing. The resulting reconstruction problem is precisely IDP.

The IDP problem becomes polynomial when the characters are directed: Benham et al. [4] studied the compatibility problem on generalized characters. Their work implies an $O(n^2m)$-time algorithm for IDP, where n and m denote the number of species and characters, respectively. Another problem related to IDP is the consensus tree problem [3, 14]. This problem calls for constructing a consensus tree from binary subtrees, and is solvable in polynomial time. One can reduce IDP to the latter problem, but the reduction itself takes $\Omega(n^2m)$ time.

Our approach to the IDP problem is graph theoretic. We first provide several graph and matrix characterizations for solvable instances of binary directed perfect phylogeny. We then reformulate IDP as a *graph sandwich* problem: The input data is recast as two nested graphs, and solving IDP is shown to be equivalent to finding a graph of a particular type "sandwiched" between them. This formulation allows us to devise efficient algorithms for IDP.

We provide two algorithms for IDP, which we call Algorithms A and B. Algorithm A has two possible implementations: deterministic and randomized. Its deterministic complexity is $O(nm + k\log^2(n+m))$, for an instance with k 1-entries in the species-characters matrix. The randomized version of Algorithm A takes $O(nm + k\log(l^2/k) + l(\log l)^3 \log\log l)$ expected time, where $l = n + m$. Algorithm B is deterministic and takes $O(l^2 \log l)$ time. For both algorithms, the improved complexity is obtained by using dynamic data structures for maintaining the connected components of a graph [23, 14, 15]. Since an $\Omega(nm)$ lower bound was shown by Gusfield for directed binary perfect phylogeny [12], our algorithms have near-optimal time complexity.

We also study the issue of multiple solutions for IDP. Often there is more than one phylogeny that is consistent with the data. When the input matrix is complete and has a solution, there is always a tree $\mathcal{T}^*$ that is *general*; i.e., it is a solution, and every other tree consistent with the data can be obtained from $\mathcal{T}^*$ by node splitting. In other words, $\mathcal{T}^*$ describes all the definite information in the data, and ambiguities (nodes

with three or more children) can be resolved by additional information. This is not always the case if the data matrix is incomplete: There may or may not be a general solution tree. In that case, using a particular solution and additional information, one can conclude that the data is inconsistent, even though the additional information may be consistent with another solution. It is thus desirable to know if a general solution exists and to generate such a solution if the answer is positive.

We provide answers to both questions. We prove that Algorithm A provides the general solution of a problem instance, if such exists. We also give an algorithm which determines if the solution $\mathcal{T}$ produced by Algorithm A is indeed general. The complexity of the latter algorithm is $O(nm + kd)$, where d denotes the maximum out-degree of $\mathcal{T}$.

The paper is organized as follows. In section 2 we provide some preliminaries, and formalize the IDP problem. In section 3 we characterize binary matrices admitting a directed perfect phylogeny, and provide the graph sandwich formulation for IDP. In section 4 we present algorithms for IDP. Finally in section 5 we analyze the generality of the solution produced by Algorithm A.

2. Preliminaries. We first specify some terminology and notation. We reserve the terms *nodes* and *branches* for trees, and use the terms *vertices* and *edges* for other graphs. Matrices are denoted by an upper-case letter, while their elements are denoted by the corresponding lower-case letter.

Let $G = (V, E)$ be a graph. We denote its set of vertices also by $V(G)$, and its set of edges also by $E(G)$. Let $\emptyset \neq V' \subseteq V$ be a subset of the vertices. The subgraph *induced* by V' is the graph $(V', E \cap (V' \times V'))$. We say that V' is *connected* in G, if V' is contained in some connected component of G. The *length* of a path in G is the number of edges along it.

Let $\mathcal{T}$ be a rooted tree with leaf set S, where branches are directed from the root towards the leaves. The *out-degree* of a node x in $\mathcal{T}$ is its number of children, and is denoted by $d(x)$. For a node x in $\mathcal{T}$ we denote the leaf set of the subtree rooted at x by $L(x)$. $L(x)$ is called a *clade* of $\mathcal{T}$. For consistency, we consider $\emptyset$ to be a clade of $\mathcal{T}$ as well, and call it the *empty clade*. $S, \emptyset$, and all singletons are called *trivial clades*. We denote by $triv(S)$ the collection of all trivial clades. Two sets are said to be *compatible* if they are either disjoint, or one of them contains the other.

OBSERVATION 1 (cf. [18]). *A collection $\mathcal{S}$ of subsets of a set S is the set of clades of some tree over S if and only if $\mathcal{S}$ contains $triv(S)$ and its subsets are pairwise compatible.*

A tree $\mathcal{T}$ is uniquely characterized by its set of clades. The transformation between a branch-node representation of a tree and a list of its clades is straightforward. Thus, we hereafter identify a tree with the set of its clades, and use the notation $S \in \mathcal{T}$ to indicate that S is a clade of $\mathcal{T}$. If $\hat{S}$ is a subset of the leaves of $\mathcal{T}$, then the subtree of $\mathcal{T}$ *induced* on $\hat{S}$ is the collection $\{\hat{S} \cap S' : S' \in \mathcal{T}\}$ (which defines a tree).

Throughout the paper we denote by $S = \{s_1, \ldots, s_n\}$ the set of all species and by $C = \{c_1, \ldots, c_m\}$ the set of all (binary) characters. For a graph K, we define $C(K) \equiv C \cap V(K)$ and $S(K) \equiv S \cap V(K)$. Let $\mathcal{B}_{n \times m}$ be a binary matrix whose rows correspond to species, each row being the character vector of its corresponding species. That is, $b_{ij} = 1$ if and only if the species s_i has the character c_j. A *phylogenetic tree for $\mathcal{B}$* is a rooted tree $\mathcal{T}$ with n leaves corresponding to the n species of S, such that each character is associated with a clade S' of $\mathcal{T}$, and the following properties are satisfied:

(1) If c_j is associated with S', then $s_i \in S'$ if and only if $b_{ij} = 1$.

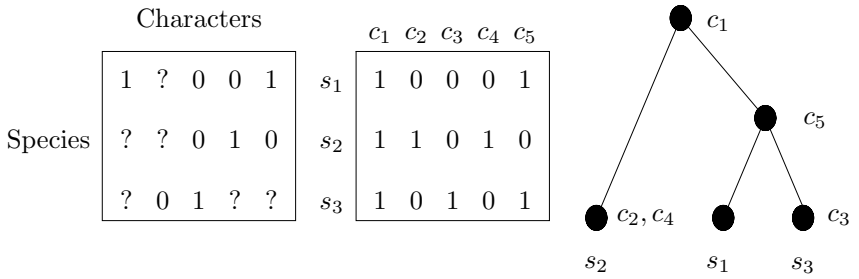


FIG. 2. *Left to right: An incomplete matrix $\mathcal{A}$, a completion $\mathcal{B}$ of $\mathcal{A}$, and a phylogenetic tree that explains $\mathcal{A}$ via $\mathcal{B}$. Each character is written to the right of its origin node.*

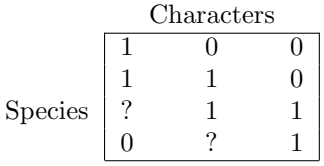


FIG. 3. *An incomplete matrix which has no phylogenetic tree although every pair of its columns has one.*

(2) Every nontrivial clade of $\mathcal{T}$ is associated with at least one character.

For a character c, the node x of $\mathcal{T}$ whose clade $L(x)$ is associated with c is called the *origin* of c with respect to $\mathcal{T}$. Characters associated with $\emptyset$ have no origin.

A $\{0, 1, ?\}$ matrix is called *incomplete*. For convenience, we consider binary matrices as incomplete. Let $\mathcal{A}_{n\times m}$ be a $\{0, 1, ?\}$ matrix in which $a_{ij} = 1$ if s_i has c_j, $a_{ij} = 0$ if s_i lacks c_j, and $a_{ij} =?$ if it is not known whether s_i has c_j. For a character c_j and a state $x \in \{0, 1, ?\}$, the *x-set* of c_j in $\mathcal{A}$ is the set of species $\{s_i \in S : a_{ij} = x\}$. c_j is called a *null character* if its 1-set is empty. For subsets $\hat{S} \subseteq S$ and $\hat{C} \subseteq C$, define $\mathcal{A}|_{\hat{S},\hat{C}}$ to be the submatrix of $\mathcal{A}$ induced on $\hat{S} \cup \hat{C}$.

A binary matrix $\mathcal{B}$ is called a *completion* of $\mathcal{A}$ if $a_{ij} \in \{b_{ij}, ?\}$ for all i, j. Thus, a completion replaces all the ?'s in $\mathcal{A}$ by zeroes and ones. If $\mathcal{B}$ has a phylogenetic tree $\mathcal{T}$, we say that $\mathcal{T}$ is a *phylogenetic tree for* $\mathcal{A}$ as well. We also say that $\mathcal{T}$ *explains* $\mathcal{A}$ *via* $\mathcal{B}$, and that $\mathcal{A}$ is *explainable*. An example of these definitions is given in Figure 2.

The following lemma, closely related to Observation 1, has been proved independently by several authors.

LEMMA 2 (cf. [18]). *A binary matrix $\mathcal{B}$ has a phylogenetic tree if and only if the* 1*-sets of every two characters are compatible.*

An analogous lemma holds for undirected characters (cf. [12]). In contrast, for incomplete matrices, even if every pair of columns has a phylogenetic tree, the full matrix might not have one. An example of such a matrix was provided in [8] for incomplete undirected characters. We provide a simpler example for incomplete *directed* characters in Figure 3. Indeed, if we consider columns 1 and 2 in the example, then the missing entry on column 1 should be completed to 1 and the one on column 2 should be completed to 0. However, in such a completion the characters on columns 2 and 3 are not compatible.

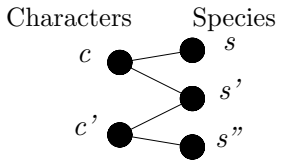


FIG. 4. *The* Σ *subgraph.*

We are now ready to state the IDP problem as follows:

Incomplete Directed Perfect Phylogeny (IDP):

Instance: An incomplete matrix $\mathcal{A}$.

Goal: Find a phylogenetic tree for $\mathcal{A}$, or determine that no such tree exists.

In section 3 we characterize complete binary matrices that admit a perfect phylogeny. In section 4 we present our algorithmic approaches for IDP.

3. Characterizations of explainable binary matrices.

3.1. Forbidden subgraph characterization. Let $\mathcal{B}$ be a species-characters binary matrix of order $n \times m$. Construct the bipartite graph $G(\mathcal{B}) = (S, C, E)$ with $E = \{(s_i, c_j) : b_{ij} = 1\}$. A Σ *subgraph* is an induced subgraph of $G(\mathcal{B})$ that includes three vertices from S, two vertices from C, and exactly four edges, forming a path of length 4 in $G(\mathcal{B})$ (see Figure 4). A bipartite graph with no induced Σ subgraph is called Σ-*free*.

The following theorem restates Lemma 2 in terms of graph theory.

THEOREM 3. *$\mathcal{B}$ has a phylogenetic tree if and only if $G(\mathcal{B})$ is Σ-free.*

COROLLARY 4. *Let $\hat{S} \subseteq S$ and $\hat{C} \subseteq C$ be subsets of the species and characters, respectively. If $\mathcal{A}$ is explainable, then so is $\mathcal{A}|_{\hat{S},\hat{C}}$.*

OBSERVATION 5. *Let $\mathcal{A}$ be a matrix explained by a tree $\mathcal{T}$ and Let $\hat{S} = L(x)$ be a clade in $\mathcal{T}$, where x is a node of $\mathcal{T}$. Then the submatrix $\mathcal{A}|_{\hat{S},C}$ is explained by the subtree of $\mathcal{T}$ rooted at x.*

For a subset $S' \subseteq S$ of species, we say that a character c is S'-*universal* in $\mathcal{B}$, if its 1-set (in $\mathcal{B}$) contains S'.

PROPOSITION 6. *If $G(\mathcal{B})$ is connected and Σ-free, then there exists a character which is S-universal in $\mathcal{B}$.*

Proof. Suppose to the contrary that $\mathcal{B}$ has no S-universal character. Consider the collection of all 1-sets of characters in $\mathcal{B}$. Let c be a character whose 1-set is maximal with respect to inclusion in this collection. Let s'' be a species which lacks c. Since $G(\mathcal{B})$ is connected, there exists a path from s'' to c in $G(\mathcal{B})$. Consider a shortest such path P. Since $G(\mathcal{B})$ is bipartite, the length of P is odd. However, P cannot be of length 1, by the choice of s''. Furthermore, if P is of length greater than 3, then its first five vertices induce a Σ subgraph, a contradiction. Thus $P = (s'', c', s', c)$ must be of length 3. By maximality of the 1-set of c, it is not contained in the 1-set of c'. Hence, there exists a species s which has the character c but lacks c'. Together with s, the vertices of P induce a Σ subgraph, as depicted in Figure 4, a contradiction. ∎

Let Ψ be a graph property. In the Ψ *sandwich problem* one is given a vertex set V and a partition of $V \times V$ into three disjoint subsets: E_0—*forbidden edges*, E_1—*mandatory edges*, and $E_?$—*optional edges*. The objective is to find a supergraph of (V, E_1) which satisfies Ψ and contains no forbidden edges. Hence, the required graph (V, F) must be "sandwiched" between (V, E_1) and $(V, E_1 \cup E_?)$. The reader is referred to articles [10, 11] for a discussion of various sandwich problems.

For the property "containing no induced Σ subgraph" (a property of bipartite graphs) the sandwich problem is defined as follows:

Σ-*free Sandwich*:

Instance: A vertex set $V = S \cup C$ with $S \cap C = \emptyset$, and a partition $E_0 \cup E_? \cup E_1$ of $S \times C$.

Goal: Find a set of edges F such that $F \supseteq E_1$, $F \cap E_0 = \emptyset$, and the graph (V, F) is Σ-free, or determine that no such set exists.

Theorem 3 motivates looking at the IDP problem with input $\mathcal{A}$ as an instance $((S, C), E_0^{\mathcal{A}}, E_?^{\mathcal{A}}, E_1^{\mathcal{A}})$ of the Σ-free sandwich problem. Here, $E_x^{\mathcal{A}} = \{(s_i, c_j) : a_{ij} = x\}$ for $x = 0, ?, 1$. In what follows, we omit the superscript $\mathcal{A}$ when it is clear from the context.

PROPOSITION 7. *The Σ-free sandwich problem is equivalent to IDP.*

Note that there is an obvious 1-1 correspondence between completions of $\mathcal{A}$ and possible solutions of the corresponding sandwich instance $((S, C), E_0, E_?, E_1)$. Hence, in what follows we refer to matrices and their corresponding sandwich instances interchangeably.

3.2. Forbidden submatrix characterizations. A binary matrix $\mathcal{B}$ is called *good* if it can be decomposed as follows:

(1) Its left $k_1 \geq 0$ columns are all ones.
(2) There exist good matrices $\mathcal{B}_1, \ldots, \mathcal{B}_l$, such that the rest (0 or more) of the columns of $\mathcal{B}$ form the block-structure illustrated in Figure 5.

A matrix $\mathcal{A}$ is *canonical* if $\mathcal{A} = [\mathcal{B}, \mathcal{C}]$, where $\mathcal{B}$ is a zero submatrix and $\mathcal{C}$ is good. We say that a matrix $\mathcal{B}$ *avoids* a matrix $\mathcal{X}$, if no submatrix of $\mathcal{B}$ is identical to $\mathcal{X}$.

THEOREM 8. *Let $\mathcal{B}$ be a binary matrix. The following are equivalent:*

1. *$\mathcal{B}$ has a phylogenetic tree.*
2. *$G(\mathcal{B})$ is Σ-free.*
3. *Every matrix obtained by permuting the rows and columns of $\mathcal{B}$ avoids the following matrix:*

$$\mathcal{Z} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

4. *There exists an ordering of the rows and columns of $\mathcal{B}$ which yields a canonical matrix.*
5. *There exists an ordering of the rows and columns of $\mathcal{B}$ so that the resulting matrix avoids the following matrices:*

$$\mathcal{X}_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \mathcal{X}_2 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathcal{X}_3 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \mathcal{X}_4 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

The reader is referred to article [17] for other problems of permuting matrices to avoid forbidden submatrices.

Proof.

1⇔2 Theorem 3.

2⇔3 Trivial.

1⇒4 Suppose $\mathcal{T}$ is a tree that explains $\mathcal{B}$. Assign to each node of $\mathcal{T}$ an index which equals its position in a preorder visit of $\mathcal{T}$. Sort the characters according to the indices of their origin nodes, letting null characters come first. Sort

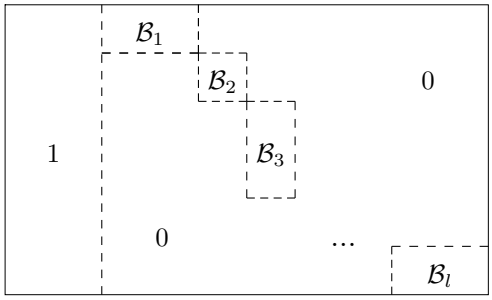


FIG. 5. *Construction of a good matrix. Each $\mathcal{B}_i$ is a good matrix. A canonical matrix is formed from it by appending columns of zeroes on the left.*

the species according to the indices of their corresponding leaves in $\mathcal{T}$. The result is a canonical matrix.

4⇒5 It is easy to verify that canonical matrices avoid $\mathcal{X}_1, \ldots, \mathcal{X}_4$.

5⇒3 Suppose to the contrary that $\mathcal{B}$ has an ordering of its rows and columns, so that rows i_1, i_2, i_3 and columns j_1, j_2 of the resulting matrix form the submatrix $\mathcal{Z}$. Consider the permutations $\theta_{row}, \theta_{col}$ of the rows and columns of $\mathcal{B}$, respectively, which yield a matrix avoiding $\mathcal{X}_1, \ldots, \mathcal{X}_4$. In this ordering, row $\theta_{row}(i_1)$ necessarily lies between rows $\theta_{row}(i_2)$ and $\theta_{row}(i_3)$ or, else, the submatrix $\mathcal{X}_4$ occurs. Suppose that $\theta_{row}(i_2) < \theta_{row}(i_3)$ and $\theta_{col}(j_1) < \theta_{col}(j_2)$; then $\mathcal{X}_3$ occurs, a contradiction. The remaining cases are similar. □

Note that a matrix which avoids $\mathcal{X}_4$ has the consecutive ones property in columns. Gusfield [12, Theorem 3] has proven that a matrix which has an *undirected* perfect phylogeny can be reordered so as to satisfy this property. In fact, for explainable binary matrices, the reordering used by Gusfield's proof essentially generates a canonical matrix. Note also that Σ-free graphs are bipartite convex as they avoid $\mathcal{X}_1$, $\mathcal{X}_2$, and $\mathcal{X}_3$ (see, e.g., [1]).

4. Algorithms for solving IDP. We now return to the problem of completing an incomplete binary matrix. Let $\mathcal{A}$ be the input matrix, and define $G(\mathcal{A}) = (S, C, E_1^{\mathcal{A}})$. For a nonempty subset $S' \subseteq S$, we say that a character is *S'-semiuniversal* in $\mathcal{A}$ if its 0-set does not intersect S'. The following lemmas motivate a divide and conquer approach to IDP, which is the basis of our algorithms for solving it.

LEMMA 9. *Let $\mathcal{A}$ be an incomplete matrix with a Σ-free completion $\mathcal{B}$. Let c be S-semiuniversal in $\mathcal{A}$. Let $\mathcal{B}'$ be the matrix obtained from $\mathcal{B}$ by setting all entries in the column of c to 1. Then $\mathcal{B}'$ is also a Σ-free completion of $\mathcal{A}$.*

Proof. Suppose to the contrary that $\{s_1, c_1, s_2, c_2, s_3\}$ induce a Σ subgraph in $G(\mathcal{B}')$. Since $G(\mathcal{B})$ is Σ-free, if follows that at least one of the Σ edges was added to $\mathcal{B}'$. But then one of c_1 and c_2 is c, a contradiction. □

LEMMA 10. *Let $\mathcal{A}$ be an incomplete matrix with a Σ-free completion $\mathcal{B}$. Let $(K_1, \ldots, K_r)$ be a partition of $S \cup C$ such that each K_i is a union of one or more connected components of $G(\mathcal{A})$. Let $\mathcal{B}'$ be the matrix obtained from $\mathcal{B}$ by setting all entries between vertices of K_i and K_j to 0, for all $i \neq j$. Then $\mathcal{B}'$ is also a Σ-free completion of $\mathcal{A}$.*

Proof. Suppose to the contrary that $\{s_1, c_1, s_2, c_2, s_3\}$ induce a Σ subgraph in $G(\mathcal{B}')$. Then one of the nonedges (s_1, c_2) or (c_1, s_3) contains one vertex from K_i and the other from K_j, for $i \neq j$. It follows that there is a path in $G(\mathcal{B}')$ between a vertex of K_i and a vertex of K_j, a contradiction. □

Alg_A$(\mathcal{A} = ((S, C), E_0, E_?, E_1))$**:**
1. **If** $|S| > 1$ then **do**:
   (a) Remove all S-semiuniversal characters and all null characters from $G(\mathcal{A})$.
   (b) If the resulting graph G' is connected, then output *False* and **halt**.
   (c) Otherwise, let $K_1, \ldots, K_r$ be the connected components of G', and let $\mathcal{A}_1, \ldots, \mathcal{A}_r$ be the corresponding submatrices of $\mathcal{A}$.
   (d) **For** $i = 1, \ldots, r$ **do**: Alg_A$(\mathcal{A}_i)$.
2. Output S.

FIG. 6. *Algorithm A for solving IDP.*

We now describe two efficient $\tilde{O}(nm)$-time algorithms for solving IDP.

4.1. Algorithm A. Algorithm A is described in Figure 6. The algorithm outputs the set of nonempty clades of a tree explaining $\mathcal{A}$, or outputs *False* if no such tree exists. The algorithm is recursive and is initially called with Alg_A$(\mathcal{A})$.

THEOREM 11. *Algorithm A correctly solves IDP.*

Proof. Suppose that the algorithm outputs *False*. Then there exists a recursive call Alg_A$(\mathcal{A}')$ in which the graph $G' = (S', C', E')$, obtained in Step 1b (see Figure 6), was found to be connected. Suppose to the contrary that $\mathcal{A}$ has a phylogenetic tree. Then by Corollary 4, there exists some edge set F^*, which solves $\mathcal{A}'$. The graph $G^* = (S', C', F^*)$ is connected and by Theorem 3, it is also Σ-free. Therefore, by Proposition 6 there exists an S'-universal character in G^*. That character must be S'-semiuniversal in $\mathcal{A}'$. By Algorithm A this vertex should have been removed at Step 1a, a contradiction.

To prove the other direction, we will show that if the algorithm outputs a collection $\mathcal{T}' = \{S_1, \ldots, S_l\}$ of sets, then $\mathcal{T} = \mathcal{T}' \cup \{\emptyset\}$ is a tree which explains $\mathcal{A}$. We first prove that the collection $\mathcal{T}$ of sets is pairwise compatible, implying by Observation 1 that $\mathcal{T}$ is a tree. Associate with each S_i the recursive call Alg_A$(\mathcal{A}_i)$ at which it was output. Observe that each such call makes recursive calls associated with disjoint subsets of S_i. By induction, it follows that $S_i \subseteq S_j$ if and only if the recursive call associated with S_i is nested within the one associated with S_j. Otherwise, $S_i \cap S_j = \emptyset$. Hence, $S_1, \ldots, S_l$ are pairwise compatible and, thus, $\mathcal{T}$ is a tree.

It remains to show that $\mathcal{T}$ is a phylogenetic tree for $\mathcal{A}$. Associate each null character with the empty clade. Each other character $\hat{c}$ is removed at Step 1a only once in the course of the algorithm, during some recursive call Alg_A$(\hat{\mathcal{A}})$. Associate $\hat{c}$ with the clade $\hat{S}$ which was output at that recursive call. Observe that each nontrivial clade $\hat{S} \in \mathcal{T}$ is associated with at least one character. Finally, define a binary matrix $\mathcal{B}_{n \times m}$ with $b_{sc} = 1$ if and only if s belongs to the clade S_c associated with c. Since $a_{sc} \neq 1$ for all $s \notin S_c$ and $a_{sc} \neq 0$ for all $s \in S_c$, $\mathcal{B}$ is a completion of $\mathcal{A}$. The claim follows. ∎

Let $h \leq \min\{m, n\}$ be the height of the reconstructed tree. Each recursive call increases the height of the output tree by at most one. The work at each level of the tree requires (1) Finding semiuniversal vertices and (2) finding connected components in disjoint graphs whose total number of edges is at most mn. Hence, the total work is $O(mn)$ per level, and a naive implementation requires $O(hmn)$ time. We give a faster implementation below.

THEOREM 12. *Algorithm A has a deterministic implementation which takes* $O(nm+|E_1|\log^2(n+m))$ *time, and a randomized implementation which takes* $O(nm+|E_1|\log(l^2/|E_1|)+l(\log l)^3\log\log l)$ *expected time, where* $l=n+m$.

Proof. For the complexity proof we give an alternative, nonrecursive implementation of Algorithm A, shown in Figure 7. This iterative version mimics the recursive one, but traverses the tree of recursive calls in a breadth first manner, rather than a depth first manner. Consequently, the implementation deals with a single graph, rather than a different graph per each recursive call. The reduction in complexity is primarily due to the use of an efficient dynamic data structure for graph connectivity. The data structure maintains the connected components of the graph while edge deletions occur.

We now analyze the running time of this implementation. Step 1 takes $O(nm)$ time. Each iteration of the "while" loop (Step 2) splits the (potential) clades added in the previous one. Thus, Algorithm A performs one iteration of this type per each level of the tree returned, and at most h iterations.

Step 2b requires explicitly computing the connected components of G. Both data structures that we use for storing the connected components of G (see below) maintain a spanning tree for each connected component of G, and allow computing the connected components in $O(n+m)$ time per iteration, or $O(h(m+n))=O(nm)$ time in total.

The loop of Step 2c is performed at most $\min\{2n-1,m\}$ times altogether, as in each (successful) iteration at least one character is removed from G (Step 2cvii), and at least one clade is added to $\mathcal{T}$. Thus, Step 2ci takes $O(\min\{n,m\})$ time altogether, and Step 2cii takes $O(nm)$ time in total. Step 2ciii takes $O(nm)$ time in total, as it considers each species-character pair only once throughout the execution of the algorithm.

In order to analyze the complexity of Step 2civ, observe that the following invariants hold in this step for each character $c\in C(K_i)$:

- $d_c^?=|\{(s,c)\in E_?|s\in S(K_i)\}|$, as guaranteed by Step 2ciii.
- $d_c^1=|\{(s,c)\in E_1|s\in S(K_i)\}|=|\{(s,c)\in E_1|s\in S\}|$, as initialized in Step 1b, since species are never removed, and each species adjacent to c must be in its connected component until c is removed.

Given $d_c^1, d_c^?$ and $|S(K_i)|$, one can check in $O(1)$ time whether c is $S(K_i)$-semiuniversal, and thus Step 2civ takes $O(|C(K_i)|)$ time, or $O(hm)$ time in total.

Since each set added to $\mathcal{T}$ in Step 2cvi corresponds to at least one character, and each character is associated with exactly one such set, updating $\mathcal{T}$ requires $O(nm)$ time in total. This also implies an $O(nm)$ bound on the size of the output produced in Step 3.

It remains to discuss the cost of the dynamic data structure, which is charged for Step 2cvii. Using the dynamic algorithm of [15], the connected components of G can be maintained during $|E_1|$ edge deletions at a total cost of $O(|E_1|\log^2(n+m))$ time spent in Step 2cvii. Alternatively, using the Las Vegas type randomized algorithm of [23] for decremental dynamic connectivity, the edge deletions can be supported in $O(|E_1|\log(l^2/|E_1|)+l(\log l)^3\log\log l)$ expected time. The complexity follows. ∎

4.2. Algorithm B. We now describe another deterministic algorithm for IDP, which is faster than Algorithm A whenever $|E_1|=\omega((n+m)^2/\log(n+m))$. Algorithm B uses the dynamic-connectivity data structure of [14], which supports deletion of batches of edges from a graph, while detecting after each batch one of the new connected components in the resulting graph (if new components were formed).

Alg_A_fast$(\mathcal{A} = ((S, C), E_0, E_?, E_1))$**:**

1. Initialize:
   (a) Set $t \leftarrow 0$, $\mathcal{K}^0 \leftarrow \{S \cup C\}$, $G \leftarrow G(\mathcal{A})$, $\mathcal{T} \leftarrow triv(S)$.
   (b) **For** each character c, and $i \in \{1, ?\}$ **do:**
   Set $d_c^i \leftarrow |\{s \in S | (s, c) \in E_i\}|$.
   (c) Remove all S-semiuniversal and all null characters from G.
   (d) Initialize a data structure for maintaining the connected components of G.
2. **While** $E(G) \neq \emptyset$ **do**:
   (a) Increment t.
   (b) Explicitly compute the set $\mathcal{K}^t$ of connected components $K_1, \ldots, K_r$ of G.
   (c) **For** each connected component $K_i \in \mathcal{K}^t$ such that $|E(K_i)| \geq 1$ **do**:
      i. Pick any character $c' \in C(K_i)$.
      ii. Compute $S' = S(K') \setminus S(K_i)$, where K' is the component in $\mathcal{K}^{t-1}$ which contains c'.
      iii. **For** each species-character pair $(s, c) \in S' \times C(K_i)$ **do**:
      If $(s, c) \in E_?$ **then** decrement $d_c^?$.
      iv. Compute the set U of all characters in K_i which are $S(K_i)$-semiuniversal in $\mathcal{A}$.
      v. **If** $U = \emptyset$, **then** output *False* and **halt**.
      vi. Set $\mathcal{T} \leftarrow \mathcal{T} \cup \{S(K_i)\}$.
      vii. Remove U from G and update the data structure of connected components accordingly.
3. Output $\mathcal{T}$.

FIG. 7. *An iterative presentation of Algorithm A.*

Algorithm B is described in Figure 8. For an instance $\mathcal{A}$ it outputs the nonempty clades of a tree explaining $\mathcal{A}$ (except possibly the root clade, if it has no matching character), or *False* if no such tree exists. It is initially called with Alg_B($\mathcal{A}$).

THEOREM 13. *Algorithm B correctly solves IDP in* $O((n + m)^2 \log(n + m))$ *deterministic time.*

Proof. Correctness. We prove correctness by induction on the problem size. If G' is connected (at Step 2c), then by Proposition 6 $\mathcal{A}$ has no phylogenetic tree, and indeed the algorithm outputs *False*. Otherwise, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the subinstances induced on K and $K' = V(G') \setminus K$, respectively, as detected in Step 2b. If $\mathcal{A}$ has a phylogenetic tree, then by Corollary 4 so do $\mathcal{A}_1$ and $\mathcal{A}_2$. On the other hand, let $\mathcal{T}_1, \mathcal{T}_2$ be phylogenetic trees for $\mathcal{A}_1, \mathcal{A}_2$, respectively. Note that by definition, $\mathcal{T}_2$ must contain the trivial clade $S(K')$, which is not necessarily a clade in a phylogenetic tree for $\mathcal{A}$ (if K' has no semiuniversal character). To remedy that, define $\mathcal{T}_2' = \mathcal{T}_2$ if the algorithm outputs $S(K')$, and $\mathcal{T}_2' = \mathcal{T}_2 \setminus \{S(K')\}$ otherwise. Then $\mathcal{T}_1 \cup \mathcal{T}_2' \cup \{S\}$ is a phylogenetic tree for $\mathcal{A}$.

Complexity. The data structure of [14] dynamically maintains a graph $H = (V, E)$ through batches of edge deletions, with each batch followed by a query for a newly created connected component in the resulting graph. If we denote by b_0 the number of batches which do not result in a new component, then as shown in [14], the total cost of answering the queries and performing the batch deletions, if eventually all edges are deleted, is $O(|V|^2 \log |V| + b_0 \min\{|V|^2, |E| \log |V|\})$.

Alg_B$(\mathcal{A} = ((S, C), E_0, E_?, E_1))$**:**

1. **If** $|S| = 1$ or $G(\mathcal{A})$ has an S-semiuniversal vertex, **then** output S.
2. **If** $|S| > 1$, **then** do:
   (a) Remove all S-semiuniversal characters and all null characters from $G(\mathcal{A})$.
   (b) **If** the resulting graph G' contains a new connected component K, **then** do:
      i. Let $\mathcal{A}_1, \mathcal{A}_2$ be the submatrices of $\mathcal{A}$ induced on $V(K)$ and $V(G') \setminus V(K)$, respectively.
      ii. **For** $i = 1, 2$ **do**: Alg_B$(\mathcal{A}_i)$.
   (c) **Else** output *False* and **halt**.

FIG. 8. *Algorithm B for solving IDP.*

	Characters			
Species	1	?	0	0
	1	1	?	?
	?	1	1	?
	?	?	1	1
	?	0	?	1

A

	Characters			
Species	1	0	0	0
	1	1	1	1
	1	1	1	1
	1	1	1	1
	1	0	1	1

B

FIG. 9. *A counterexample to the greedy approach. A: The input matrix. B: A solution.*

We use this data structure to maintain $G(\mathcal{A})$ during all the recursive calls. As $b_0 = 1$ (since in case no new component is formed the algorithm outputs *False* and halts) and $|V| = n+m$, the total cost is $O((m+n)^2 \log(n+m))$ time. This expression dominates the complexity, as finding the semiuniversal vertices at each recursive call costs in total only $O(nm)$ time (see proof of Theorem 12). □

We remark that an $\Omega(nm)$-time lower bound for (undirected) binary perfect phylogeny was proved by Gusfield [12]. A closer look at Gusfield's proof reveals that it applies, as is, also to the directed case. As IDP generalizes directed binary perfect phylogeny, any algorithm for this problem would require $\Omega(nm)$ time.

4.3. Greedy approach fails. We end the section by showing that a simple greedy approach to IDP fails. Let $\mathcal{A}$ be an incomplete matrix. We say that $a_{sc} =?$ is *forced* if there exists an assignment $x \in \{0, 1\}$ such that completing a_{sc} to x results in an induced Σ in the graph $(S, C, E_1^{\mathcal{A}'} \cup E_?^{\mathcal{A}'})$ corresponding to the completed matrix $\mathcal{A}'$. $\mathcal{A}$ is called *forced* if it has some forced ?-entry.

A naive greedy algorithm for IDP is as follows: At each step complete one ?-entry in the matrix. If there are no forced entries, choose any ?-entry and complete it arbitrarily. Otherwise, try to complete a forced entry. If such completion is not possible (an induced Σ is formed), report *False*.

Figure 9(A) shows an explainable instance with no forced entries. Setting the bottom-left ?-entry to 0 results in an instance which cannot be explained. A solution matrix is shown in Figure 9(B).

5. Determining the generality of the solution. A "yes" instance of IDP may have several distinct phylogenetic trees as solutions. These trees may be related in the following way: We say that a tree $\mathcal{T}$ *generalizes* a tree $\mathcal{T}'$, and write $\mathcal{T} \subseteq \mathcal{T}'$,

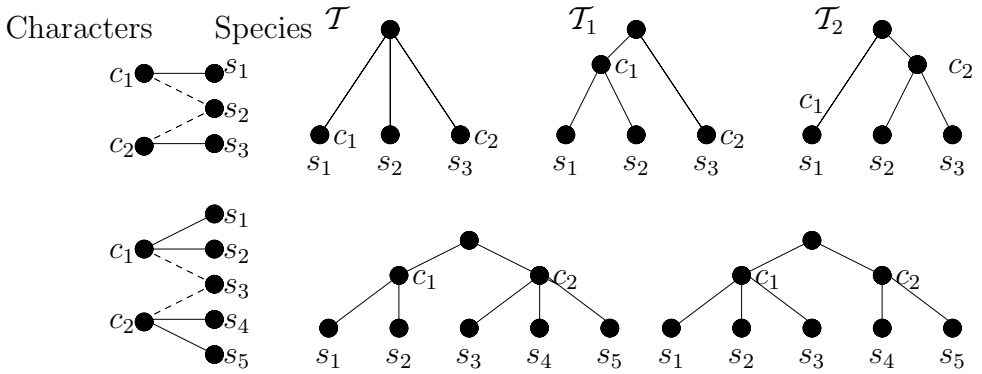


FIG. 10. *Top left: An IDP instance which has a general solution. Dashed lines denote $E_?$-edges, while solid lines denote E_1-edges. Top-right: $\mathcal{T}, \mathcal{T}_1$, and $\mathcal{T}_2$ are the possible solutions. $\mathcal{T}$ generalizes $\mathcal{T}_1$ and $\mathcal{T}_2$ (which are obtained by splitting the root node of $\mathcal{T}$), and is the general solution. Bottom left: An IDP instance which has no general solution. Bottom middle and bottom right: Two possible solutions. The only tree which generalizes both solutions is the tree composed of the trivial clades only, which is not a solution.*

if every clade of $\mathcal{T}$ is a clade of $\mathcal{T}'$; i.e., the evolutionary scenario expressed by $\mathcal{T}'$ includes all the details of the scenario expressed by $\mathcal{T}$, and possibly more. Therefore, $\mathcal{T}'$ represents a more specific scenario, and $\mathcal{T}$ represents a more general one. We say that a tree $\mathcal{T}$ is the *general solution* of an instance $\mathcal{A}$, if $\mathcal{T}$ explains $\mathcal{A}$ and generalizes every other tree which explains $\mathcal{A}$. Figure 10 demonstrates the definitions and also gives an example of an instance that has no general solution.

We give in this section a characterization of IDP instances that admit a general solution. We prove that whenever a general solution exists, Algorithm A finds it. We also provide an algorithm to determine whether the solution tree $\mathcal{T}$ returned by Algorithm A is general. The complexity of the latter algorithm is shown to be $O(mn + |E_1|d)$, where d is the maximum out-degree in $\mathcal{T}$.

This notation is used in what follows. Let $\mathcal{A}$ be an incomplete matrix and let $\hat{S} \subseteq S$. We denote by $W_{\mathcal{A}}(\hat{S})$ the set of $\hat{S}$-semiuniversal characters in $\mathcal{A}$. Note that if $\mathcal{A}$ is binary, then $W_{\mathcal{A}}(\hat{S})$ is its set of $\hat{S}$-universal characters. We now define the operator $\widetilde{\ }$ on incomplete matrices: We denote by $\widetilde{\mathcal{A}}$ the submatrix $\mathcal{A}|_{S,C\setminus W_{\mathcal{A}}(S)}$ of $\mathcal{A}$. In particular, $G(\widetilde{\mathcal{A}})$ is the graph produced from $G(\mathcal{A})$ by removing its set of S-semiuniversal characters. A species set $\emptyset \neq S' \subseteq S$ is said to be *connected* in a graph G, if S' is contained in some connected component of G.

LEMMA 14. *Let $\mathcal{T}$ be the general solution for an instance $\mathcal{A}$ of IDP. Let $S' = L(x)$ be a clade of $\mathcal{T}$, corresponding to some node x. Let $\mathcal{T}'$ be the subtree of $\mathcal{T}$ rooted at x, and let $\mathcal{A}'$ be the instance induced on $S' \cup C$. Then $\mathcal{T}'$ is the general solution for $\mathcal{A}'$.*

Proof. By Observation 5, $\mathcal{T}'$ explains $\mathcal{A}'$. Suppose that $\mathcal{T}''$ also explains $\mathcal{A}'$ and $\mathcal{T}' \not\subseteq \mathcal{T}''$. Then $\hat{\mathcal{T}} = (\mathcal{T} \setminus \mathcal{T}') \cup \mathcal{T}''$ explains $\mathcal{A}$, and $\mathcal{T} \not\subseteq \hat{\mathcal{T}}$, a contradiction. ∎

A nonempty clade of a tree is called *maximal* if the only clade that properly contains it is S.

LEMMA 15. *Let $\mathcal{T}$ be a phylogenetic tree for a binary matrix $\mathcal{B}$. A nonempty clade S' of $\mathcal{T}$ is maximal if and only if S' is the species set of some connected component of $G(\widetilde{B})$.*

Proof. Suppose that S' is a maximal clade of $\mathcal{T}$. We first claim that S' is contained in some connected component K of $G(\widetilde{B})$. If $|S'| = 1$ this trivially holds. If $|S'| > 1$,

let c be a character associated with S'. c is adjacent to all vertices in S' and to no other vertex. Hence, c is not S-universal, implying that all the edges $\{(c,s) : s \in S'\}$ are present in $G(\widetilde{B})$. This proves the claim. It remains to show that $S(K) = S'$. Suppose $S(K) \supset S'$. In particular, $|S(K)| > 1$. By Proposition 6, there exists a character c' in $G(\widetilde{B})$ whose 1-set is $S(K)$. Hence, $S(K)$ must be a clade of $\mathcal{T}$ which is associated with c', contradicting the maximality of S'.

To prove the converse, let S' be the species set of some connected component K of $G(\widetilde{B})$. We first claim that S' is a clade. If $|S'| = 1$, S' is a trivial clade. Otherwise, by Proposition 6 there exists an S'-universal character c' in $G(\widetilde{B})$. Since K is a connected component, c' has no neighbors in $S \setminus S'$. Hence, S' must be a clade in $\mathcal{T}$. Suppose to the contrary that S' is not maximal; then it is properly contained in a maximal clade S'', which by the previous direction is the species set of K, a contradiction. ∎

THEOREM 16. *Algorithm A produces the general solution for every IDP instance that has one.*

Proof. Let $\mathcal{A}$ be an instance of IDP for which there exists a general solution $\mathcal{T}^*$. Let $\mathcal{T}_{alg}$ be the solution tree produced by Algorithm A. By definition $\mathcal{T}^* \subseteq \mathcal{T}_{alg}$. Suppose to the contrary that $\mathcal{T}^* \neq \mathcal{T}_{alg}$. Let S' be the largest clade reported by Algorithm A, which is not a clade of $\mathcal{T}^*$ (S' must be nontrivial), and let S'' be the smallest clade in $\mathcal{T}_{alg}$ which properly contains S'. Let $\mathcal{A}'$ be the instance induced on $S'' \cup C$. By Observation 5, $\mathcal{A}'$ is explained by the corresponding subtrees $\mathcal{T}'_{alg}$ of $\mathcal{T}_{alg}$ and $\mathcal{T}'^*$ of $\mathcal{T}^*$. By Lemma 14, $\mathcal{T}'^*$ is the general solution of $\mathcal{A}'$. Due to the recursive nature of Algorithm A, it produces $\mathcal{T}'_{alg}$ when invoked with input $\mathcal{A}'$. Thus, without loss of generality, one can assume that $S'' = S$ and S' is a maximal clade of $\mathcal{T}_{alg}$.

Suppose that $\mathcal{T}^*$ explains $\mathcal{A}$ via a completion $\mathcal{B}^*$, and let $G^* = G(\mathcal{B}^*)$. Since S' is a maximal clade, it is reported during a second level call of Alg_A($\cdot$) (the call at the first level reports the trivial clade S). Hence, it must be the species set of some connected component K in $G(\widetilde{\mathcal{A}})$. Since every S-universal character in G^* is S-semiuniversal in $\mathcal{A}$, S' is contained in some connected component K^* of $G(\widetilde{B^*})$. Denote $S^* \equiv S(K^*)$. By Lemma 15, S^* is a maximal clade of $\mathcal{T}^*$. Since $S' \notin \mathcal{T}^*$, we have $S' \neq S^*$, and therefore, $S^* \supset S'$. But $\mathcal{T}^* \subseteq \mathcal{T}_{alg}$, implying that S^* is also a nontrivial clade of $\mathcal{T}_{alg}$, in contradiction to the maximality of S'. ∎

We now characterize IDP instances for which a general solution exists. Let $\mathcal{A}$ be a "yes" instance of IDP. Consider a recursive call Alg_A($\mathcal{A}'$) nested within Alg_A($\mathcal{A}$), where $\mathcal{A}' = \mathcal{A}|_{C',S'}$. Let $K_1, \ldots, K_r$ be the connected components of $G(\widetilde{\mathcal{A}'})$, computed in Step 1c. Observe that $S(K_1), \ldots, S(K_r)$ are clades to be reported by recursive calls launched during Alg_A($\mathcal{A}'$). A set U of characters is said to be (K_i, K_j)-*critical* if characters in U are both $S(K_i)$-semiuniversal and $S(K_j)$-semiuniversal in $\mathcal{A}'$, and removing U from $G(\widetilde{\mathcal{A}'})$ disconnects $S(K_i)$. Note that by definition of U, $U \subseteq W_{\mathcal{A}'}(S(K_i))$, and $a'_{sc} =?$ for all $c \in U, s \in S(K_j)$. A clade $S(K_i)$ is called *optional* (with respect to $\mathcal{A}'$) if $r \geq 3$ and there exists a (K_i, K_j)-critical set for some index $j \neq i$. If $S(K_i)$ is not optional we say it is *mandatory*. In the example of Figure 10 (bottom), let $K_1 = \{s_1, s_2, c_1\}$, $K_2 = \{s_3\}$, and $K_3 = \{s_4, s_5, c_2\}$. The set $U = \{c_1\}$ is (K_1, K_2)-critical, so $S(K_1) = \{s_1, s_2\}$ is optional. In contrast, in Figure 10 (top) no clade is optional.

THEOREM 17. *The tree produced by Algorithm A is the general solution if and only if all its clades are mandatory.*

Proof. ⇒ Suppose that $\mathcal{T}_{alg}$ is the general solution of an instance $\mathcal{A}$. Suppose to the contrary that it contains an optional clade. Without loss of generality, assume it is maximal; i.e., during the recursive call Alg_A($\mathcal{A}$), $G' = G(\widetilde{\mathcal{A}})$ has $r \geq 3$ connected

components, $K_1, \ldots, K_r$, and there exists a (K_i, K_j)-critical set U (for some $1 \leq i \neq j \leq r$). Let $\mathcal{A}_i, \mathcal{A}_j$ and $\mathcal{A}_{ij}$ be the subinstances induced on K_i, K_j and $K_i \cup K_j$, respectively. Consider the tree $\mathcal{T}'$ which is produced by a small modification to the execution of Alg_A($\mathcal{A}$): Instead of recursively invoking Alg_A($\mathcal{A}_i$) and Alg_A($\mathcal{A}_j$), call Alg_A($\mathcal{A}_{ij}$). Then $\mathcal{T}'$ is a phylogenetic tree which explains $\mathcal{A}$ and includes the clade $S(K_i \cup K_j)$. Since removing U from $G(\widetilde{\mathcal{A}})$ disconnects $S(K_i)$, $|S(K_i)| \geq 2$ so $S(K_i)$ is nontrivial. Moreover, $S(K_i)$ is not a clade of $\mathcal{T}'$ for the same reason. Hence, $\mathcal{T}'$ does not contain all clades of $\mathcal{T}_{alg}$, in contradiction to the generality of $\mathcal{T}_{alg}$.

$\Leftarrow$ Suppose that $\mathcal{T}_{alg}$ is not the general solution of an instance $\mathcal{A}$; i.e., there exists a solution $\mathcal{T}^*$ of $\mathcal{A}$ such that $\mathcal{T}_{alg} \not\subseteq \mathcal{T}^*$. We shall prove the existence of an optional clade in $\mathcal{T}_{alg}$. (The reader is referred to the example in Figure 13 for notation and intuition. The example follows the steps of the proof, leading to the identification of an optional clade.) Let $\mathcal{B}^*$ be a completion of $\mathcal{A}$ which is explained by $\mathcal{T}^*$, and denote $G^* = G(\mathcal{B}^*)$. Let $S' \in \mathcal{T}_{alg} \setminus \mathcal{T}^*$ be the largest clade reported by Algorithm A which is not a clade of $\mathcal{T}^*$. Without loss of generality (as argued in the proof of Theorem 16), S' is a maximal clade of $\mathcal{T}_{alg}$, and we let $S' = S(K_1)$, where $K_1, \ldots, K_r$ are the connected components of $G(\widetilde{\mathcal{A}})$.

Observe that a binary matrix has at most one phylogenetic tree. Thus, an application of Algorithm A to $\mathcal{B}^*$ necessarily outputs $\mathcal{T}^*$. Consider such an application, and let $\{S_i^*\}_{i=1}^t$ be the nested set of reported clades in $\mathcal{T}^*$ which contain S': $S = S_1^* \supset \cdots \supset S_t^* \supset S'$ (see Figure 11). For each $i = 1, \ldots, t$, let $\mathcal{B}_i^*$ be the instance invoked in the recursive call which reports S_i^*, and let H_i^* be the graph $G(\widetilde{\mathcal{B}_i^*})$, computed in Step 1a of that recursive call. Let C_i^* be the set of characters in H_i^*. Equivalently, C_i^* is the set of characters in $\mathcal{B}_i^*$ whose 1-set is nonempty and is properly contained in S_i^*. Furthermore, define H_i to be the subgraph of $G(\mathcal{A})$ induced on $S_i^* \cup C_i^*$. Observe that H_i^* is the subgraph of G^* induced on the same vertex set. Since G^* is a supergraph of $G(\mathcal{A})$, each H_i^* is a supergraph of H_i.

Claim 18. *S' is disconnected in H_t^*, and therefore also in H_t.*

Proof. Suppose to the contrary that S' is contained in some connected component K^* of H_t^*. K^* is thus computed during the tth recursive call (with argument $\mathcal{B}_t^*$), and $S(K^*)$ is reported as a clade in $\mathcal{T}^*$ by a nested recursive call. Therefore, $S_t^* \supset S(K^*) \supset S'$, where the first proper containment follows from the fact that H_t^* is disconnected, and the second from the assumption that S' is not a clade of $\mathcal{T}^*$. Hence, we arrive at a contradiction to the minimality of S_t^*. ∎

We now return to the proof of Theorem 17. Recall that S' is connected in $H_1 = G(\widetilde{A})$. Thus, the previous claim implies that $t > 1$. Let K_p be a connected component of $G(\widetilde{\mathcal{A}})$ such that $S(K_p) \subseteq S \setminus S_2^*$ (see Figure 11). Let l be the minimal index such that there exists some connected component K_i of $G(\widetilde{\mathcal{A}})$ for which $S(K_i)$ is disconnected in H_l. l is properly defined as $S(K_1) = S'$ is disconnected in H_t. $l > 1$, since otherwise some K_i is disconnected in H_1 and, therefore, also in its subgraph $G(\widetilde{\mathcal{A}})$, in contradiction to the definition of $K_1, \ldots, K_r$.

By minimality of l, $S_l^* \supseteq S(K_i)$. Also, $S_l^* \supseteq S_t^* \supset S' = S(K_1)$, so $S_l^* \neq S(K_i)$. We now claim that there exists some connected component K_j of $G(\widetilde{\mathcal{A}})$, $j \neq i$, such that $S(K_j) \subseteq S_l^*$. Indeed, if $i \neq 1$, then $j = 1$. If $i = 1$, then $l = t$ (by an argument similar to that in the proof of Claim 18), and since $S_l^* \setminus S'$ is nonempty, it intersects $S(K_j)$ for some $j \neq i$. By minimality of l, $S(K_j)$ is properly contained in $S_l^* \setminus S'$.

Define $U \equiv W_{G^*}(S_l^*)$. We now prove that U is a (K_i, K_j)-critical set. By definition all characters in U are S_l^*-universal in G^*, and are thus both K_i-semiuniversal

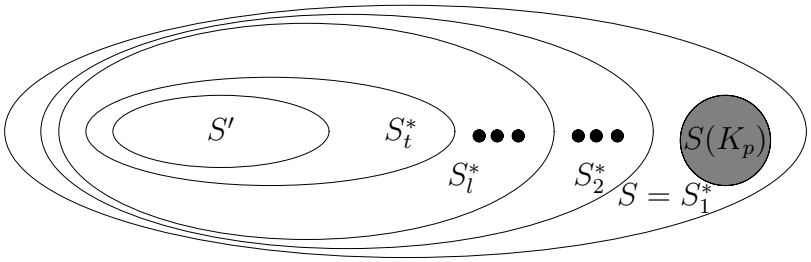


FIG. 11. *The clades* $S = S_1^* \supset S_2^* \supset \cdots \supset S_t^* \supset S'$.

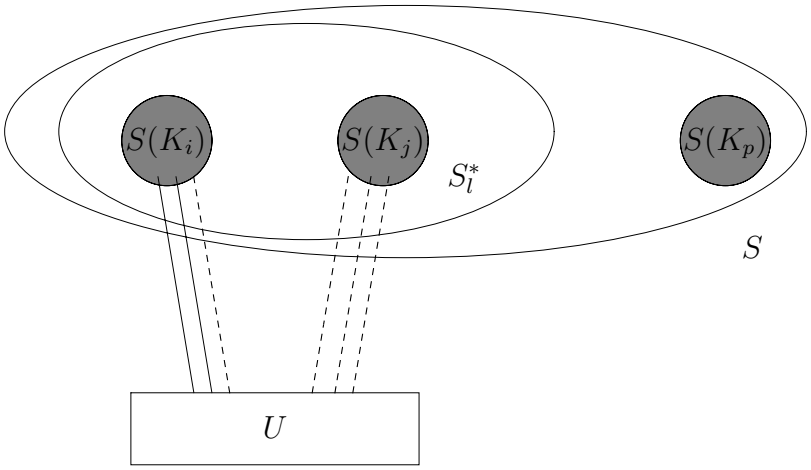


FIG. 12. $S(K_i)$, $S(K_j)$, *and* $S(K_p)$. *Note that removing* U *disconnects* $S(K_i)$.

and K_j-semiuniversal in $\mathcal{A}$. $S(K_i)$ is disconnected in $H_l = G(\mathcal{A}|_{C_l^*,S_l^*})$. Since K_i is a connected component of $G(\widetilde{\mathcal{A}})$, $S(K_i)$ is disconnected in $G(\mathcal{A}|_{C_l^*,S})$, implying that U is a (K_i, K_j)-critical set. Also, K_i, K_j, and K_p are distinct, implying that $r \geq 3$ (see Figure 12). In conclusion, U demonstrates that $S(K_i)$ is optional. □

The characterization of Theorem 17 leads to an efficient algorithm for determining whether a solution $\mathcal{T}_{alg}$ produced by Algorithm A is general.

THEOREM 19. *There is an* $O(nm + |E_1|d)$*-time algorithm to determine if a given solution* $\mathcal{T}_{alg}$ *is general, where* d *is the maximum out-degree in* $\mathcal{T}_{alg}$.

Proof. The algorithm simply traverses $\mathcal{T}_{alg}$ bottom-up, searching for optional clades. For each internal node x visited, whose children are $y_1, \ldots, y_{d(x)}$, the algorithm checks whether any of the clades $L(y_1), \ldots, L(y_{d(x)})$ is optional. If an optional clade is found the algorithm outputs *False*. Correctness follows from Theorem 17.

We show how to efficiently check whether a clade $L(y_i)$ is optional. If $d(x) = 2$, or y_i is a leaf, then certainly $L(y_i)$ is mandatory. Otherwise, let U_i be the set of characters whose origin (in $\mathcal{T}_{alg}$) is y_i. Let U_j^i denote the set of characters in U_i which are $L(y_j)$-semiuniversal, for $j \neq i$. The computation of U_j^i for all i and j takes $O(nm)$ time in total, since for each character c and species s we check at most once whether $(s, c) \in E_?^{\mathcal{A}}$, for an input instance $\mathcal{A}$.

It remains to show how to efficiently check whether for some j, U_j^i disconnects $L(y_i)$ in the appropriate subgraph encountered during the execution of Algorithm A. To this end, we define an auxiliary bipartite graph H^i whose set of vertices is $W_i \cup U_i$,

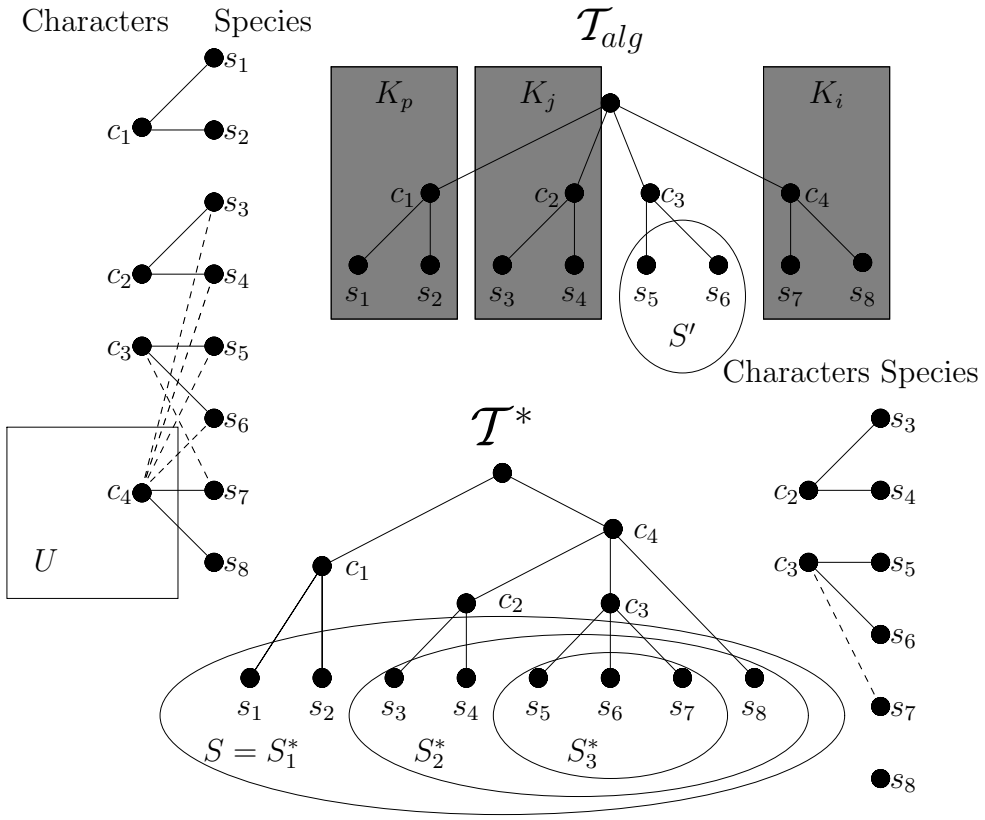


FIG. 13. *An example demonstrating the proof of the "if" part of Theorem* 17*, using the notation in the proof. Left: A graphical representation of an input instance* $\mathcal{A}$*. Dashed lines denote* $E_?$*-edges, while solid lines denote* E_1*-edges. Top right: The tree* $\mathcal{T}_{alg}$ *produced by Algorithm A. Bottom middle: A tree* $\mathcal{T}^*$ *corresponding to a completion* $\mathcal{B}^*$ *that uses all the edges in* $E_?$*. Bottom right: The graphs* H_2 *(solid edges) and* H_2^* *(solid and dashed edges).* $\mathcal{T}_{alg} \not\subseteq \mathcal{T}^*$*, and* $S' = \{s_5, s_6\}$*. There are* $t = 3$ *clades of* $\mathcal{T}^*$ *which contain* S'*:* $S_1^* = \{s_1, \ldots, s_8\}$*,* $S_2^* = \{s_3, \ldots, s_8\}$*, and* $S_3^* = \{s_5, s_6, s_7\}$*. The component* $K_p = \{c_1, s_1, s_2\}$ *has its species in* $S \setminus S_2^*$*. Since* $W_{\mathcal{A}}(S) = W_{\mathcal{B}^*}(S) = \emptyset$*,* $H_1 = G(\mathcal{A})$*. Since* $W_{\mathcal{B}^*}(S_2^*) = \{c_4\}$*, the species set of the connected component* $K_i = \{s_7, s_8, c_4\}$ *is disconnected in* H_2*, implying that* $l = 2$*. For a choice of* $K_j = \{s_3, s_4, c_2\}$*, the set* $U = \{c_4\}$ *is* (K_i, K_j)*-critical, demonstrating that* S' *is optional.*

where $W_i = \{w_1, \ldots, w_{d(y_i)}\}$ is the set of children of y_i in $\mathcal{T}_{alg}$. We include the edge (w_r, c_p) in H^i, for $w_r \in W_i, c_p \in U_i$, if $(c_p, s) \in E_1^{\mathcal{A}}$ for some species $s \in L(w_r)$. We construct for each $j \neq i$ a subgraph H_j^i of H^i induced on $W_i \cup (U_i \setminus U_j^i)$. All we need to report is whether H_j^i is connected.

For each i we construct H^i by considering all $E_1^{\mathcal{A}}$ edges connecting characters in U_i to species in $L(y_i)$. This takes $O(|E_1^{\mathcal{A}}|)$ time in total. There are $d(y_i)$ subgraphs H_j^i for every y_i. Hence, computing H_j^i for all j and determining whether each H_j^i is connected take $O\left(|E(H^i)|d(y_i)\right)$ time. Since $\sum_i |E(H^i)| \leq |E_1^{\mathcal{A}}|$, the total time complexity is $O(mn + \sum_i |E(H^i)|d(y_i)) = O(mn + |E_1^{\mathcal{A}}| \cdot \max_{v \in \mathcal{T}_{alg}} d(v))$. ∎

Acknowledgments. We thank Dan Graur for drawing our attention to this phylogenetic problem, and for helpful discussions. We thank Nati Linial for insightful comments. We thank Joe Felsenstein, Dan Gusfield, Haim Kaplan, Mike Steel, and an anonymous CPM '00 referee for referring us to helpful literature.

REFERENCES

[1] N. Abbas and L. Stewart, *Biconvex graphs: Ordering and algorithms*, Discrete Appl. Math., 103 (2000), pp. 1–19.

[2] R. Agarwala and D. Fernández-Baca, *A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed*, SIAM J. Comput., 23 (1994), pp. 1216–1224.

[3] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, *Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions*, SIAM J. Comput., 10 (1981), pp. 405–421.

[4] C. Benham, S. Kannan, M. Paterson, and T. Warnow, *Hen's teeth and whale's feet: Generalized characters and their compatibility*, J. Comput. Biology, 2 (1995), pp. 515–525.

[5] H. L. Bodlaender, M. R. Fellows, M. T. Hallett, T. Wareham, and T. Warnow, *The hardness of perfect phylogeny, feasible register assignment and other problems on thin colored graphs*, Theoret. Comput. Sci., 244 (2000), pp. 167–188.

[6] J. H. Camin and R. R. Sokal, *A method for deducing branching sequences in phylogeny*, Evolution, 19 (1965), pp. 409–414.

[7] L. Dollo, *Le lois de l'évolution*, Bulletin de la Societé Belge de Géologie de Paléontologie et d'Hydrologie, 7 (1893), pp. 164–167.

[8] J. Felsenstein, *Inferring Phylogenies*, Sinauer Associates, Sunderland, MA, 2003.

[9] L. R. Foulds and R. L. Graham, *The Steiner problem in phylogeny is NP-complete*, Adv. in Appl. Math., 3 (1982), pp. 43–49.

[10] M. C. Golumbic, *Matrix sandwich problems*, Linear Algebra Appl., 277 (1998), pp. 239–251.

[11] M. C. Golumbic, H. Kaplan, and R. Shamir, *Graph sandwich problems*, J. Algorithms, 19 (1995), pp. 449–473.

[12] D. Gusfield, *Efficient algorithms for inferring evolutionary trees*, Networks, 21 (1991), pp. 19–28.

[13] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, Cambridge, UK, 1997.

[14] M. Henzinger, V. King, and T. Warnow, *Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology*, Algorithmica, 24 (1999), pp. 1–13.

[15] J. Holm, K. de Lichtenberg, and M. Thorup, *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree,* 2*-edge, and biconnectivity*, J. ACM, 48 (2001), pp. 723–760.

[16] S. Kannan and T. Warnow, *A fast algorithm for the computation and enumeration of perfect phylogenies*, SIAM J. Comput., 26 (1997), pp. 1749–1763.

[17] B. Klinz, R. Rudolf, and G. J. Woeginger, *Permuting matrices to avoid forbidden submatrices*, Discrete Appl. Math., 60 (1995), pp. 223–248.

[18] C. A. Meecham and G. F. Estabrook, *Compatibility methods in systematics*, Ann. Rev. Ecol. and Syst., 16 (1985), pp. 431–446.

[19] M. Nikaido, A. P. Rooney, and N. Okada, *Phylogenetic relationships among cetartiodactyls based on insertions of short and long interspersed elements: Hippopotamuses are the closest extant relatives of whales*, Proc. Natl. Acad. Sci. USA, 96 (1999), pp. 10261–10266.

[20] W. J. Le Quesne, *The uniquely evolved character concept and its cladistic application*, Systematic Zoology, 23 (1974), pp. 513–517.

[21] M. A. Steel, *The complexity of reconstructing trees from qualitative characters and subtrees*, J. Classification, 9 (1992), pp. 91–116.

[22] D. L. Swofford, *PAUP, Phylogenetic Analysis Using Parsimony (and Other Methods)*, Version 4, Sinauer Associates, Sunderland, MA, 1998; also available online from http://paup.csit.fsu.edu/.

[23] M. Thorup, *Decremental dynamic connectivity*, J. Algorithms, 33 (1999), pp. 229–243.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 608–631

$(1+\epsilon,\beta)$-SPANNER CONSTRUCTIONS FOR GENERAL GRAPHS*

MICHAEL ELKIN† AND DAVID PELEG‡

Abstract. An (α,β)-*spanner* of a graph G is a subgraph H such that $dist_H(u,w) \leq \alpha \cdot distt_G(u,w)+\beta$ for every pair of vertices u,w, where $dist_{G'}(u,w)$ denotes the distance between two vertices u and v in G'. It is known that every graph G has a polynomially constructible $(2\kappa-1,0)$-spanner (also known as *multiplicative* $(2\kappa-1)$-spanner) of size $O(n^{1+1/\kappa})$ for every integer $\kappa \geq 1$, and a polynomially constructible $(1,2)$-spanner (also known as *additive* 2-spanner) of size $\tilde{O}(n^{3/2})$. This paper explores hybrid spanner constructions (involving both multiplicative and additive factors) for general graphs and shows that the multiplicative factor can be made *arbitrarily close to* 1 while keeping the spanner size arbitrarily close to $O(n)$, at the cost of allowing the additive term to be a sufficiently large constant. More formally, we show that for any constant $\epsilon,\lambda > 0$ there exists a constant $\beta = \beta(\epsilon,\lambda)$ such that for every n-vertex graph G there is an efficiently constructible $(1+\epsilon,\beta)$-spanner of size $O(n^{1+\lambda})$.

Key words. spanners, graph algorithms, graph partitions

AMS subject classifications. 05C85, 05C12, 68R10

DOI. 10.1137/S0097539701393384

1. Introduction.

1.1. Motivation and previous results.

Spanners for general graphs were introduced in [15] as a tool for constructing synchronizers, and were later studied in various contexts. Generally speaking, spanners appear to be the underlying graph structure in a number of constructions in distributed systems and communication networks (cf. [13]). Spanners have turned out to be relevant also in other contexts. For instance, in the area of robotics and computational geometry, spanners have been considered in the Euclidean setting, assuming that the vertices of the graph are points in space, and the edges are line segments connecting pairs of points (cf. [5, 6, 1] and the references therein).

Intuitively, spanners can be thought of as a generalization of the concept of a spanning tree, allowing the spanning subgraph to have cycles, but aiming towards maintaining the locality properties of the network. These locality properties revolve around the notion of *stretch*, namely, the (worst) multiplicative factor by which distances increase in the network as a result of using the spanner edges alone and ignoring non-spanner edges. Formally, given an unweighted graph $G=(V,E)$, we say that the subgraph $H=(V,E')$ (where $E' \subseteq E$) is an α-*spanner* of G if $dist_H(u,w) \leq \alpha \cdot dist_G(u,w)$ for every $u,w \in V$, where $dist_{G'}(u,v)$ denotes the distance between two vertices u and v in G', namely, the minimum length of a path in G' connecting them.

There exists a well-understood tradeoff between the size of a spanner (namely, the number of edges it uses) and its stretch [14, 1, 4]. Generally speaking, for an integer parameter κ, a stretch of $O(\kappa)$ can be guaranteed by a spanner using $O(n^{1+1/\kappa})$ edges. The best known bound for the stretch is $2\kappa-1$ [1]. This bound is very close to the

*Received by the editors August 5, 2001; accepted for publication (in revised form) November 12, 2003; published electronically March 30, 2004.
http://www.siam.org/journals/sicomp/33-3/39338.html

†Department of Computer Science, Yale University, New Haven, CT 06520 (elkin@cs.yale.edu). This author's work was done at the Weizmann Institute of Science, Rehovot, Israel.

‡Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, 76100 Israel (peleg@wisdom.weizmann.ac.il). This author's work was supported in part by grants from the Israel Science Foundation and the Israel Ministry of Science and Art.

best possible in view of the lower bounds shown in [14, 1, 4], by which for every odd $\kappa \geq 3$ there exist (infinitely many) n-vertex graphs $G = (V, E)$ for which every $(\kappa - 2)$-spanner requires $\Omega(n^{1+4/3\kappa})$ edges. In fact, for 3-, 5-, and 9-spanners, even better lower bounds are known [16].

However, it is not clear a priori that the stretch must be expressed as a *multiplicative* factor. An alternative notion of *additive graph spanners* was introduced in [11, 12]. A subgraph H is an *additive β-spanner* of G if $dist_H(u, w) \leq dist_G(u, w) + \beta$ for every $u, w \in V$. While the results of [11, 12] concerned only special graph classes, like pyramids, hypercubes and multidimensional grids of degree bounded by 4, a result demonstrating the potential usefulness of this notion for general graphs was presented in [7], where it was shown that for every graph G there exists an additive 2-spanner with $\tilde{O}(n^{3/2})$ edges. Again, this is the best possible up to polylogarithmic factors given the aforementioned lower bounds. Unfortunately, this result has so far resisted attempts of extending it to values of κ greater than 2.

In this paper, we study the somewhat more general and unifying concept of (α, β)-*spanners*, also introduced in [11]. A subgraph H is an (α, β)-*spanner* of G if $dist_H(u, w) \leq \alpha \cdot dist_G(u, w) + \beta$ for every $u, w \in V$. Cast in this terminology, the above mentioned results imply that $O(n^{1+1/\kappa})$ edges suffice to construct a $(2\kappa - 1, 0)$-spanner, for any integer $\kappa \geq 1$, and that $\tilde{O}(n^{3/2})$ edges suffice to construct a $(1, 2)$-spanner.

However, intuitively it seems that a tighter bound on the behavior of spanners may be obtained. In particular, it seems plausible that the multiplicative factor of $2\kappa - 1$ is not entirely unavoidable, as it might stem in part due to a "hidden" additive term affecting mainly the stretching behavior of the spanner with respect to *nearby* vertex pairs.

The current paper not only confirms this intuition, but also shows that the multiplicative stretch can be made *arbitrarily close to* 1 by allowing the additive term to be a sufficiently large *constant*.

1.2. Our results. Our main construction establishes the existence and efficient constructibility of $(1 + \epsilon, \beta)$-spanners of size $O(\beta n^{1+1/\kappa})$ for every n-vertex graph G, where $\beta = \beta(\kappa, \epsilon)$ is constant whenever κ and ϵ are. We stress that in our construction, both the stretch and the size of the spanner can be made arbitrarily small *simultaneously*, where the tradeoff is between their values on the one hand and the value of the additive term on the other.

Note that the existence of spanners with properties as described above implies that for any constant $\epsilon, \lambda > 0$ and graph G, there exists a spanning subgraph H with $O(n^{1+\lambda})$ edges which behaves like a $(1 + \epsilon)$-spanner for "sufficiently distant" pairs of vertices. More formally, for any constant $\epsilon, \lambda > 0$ there exists a constant $\beta(\epsilon, \lambda)$ such that for any n-vertex graph $G = (V, E)$ there exists an efficiently constructible spanning subgraph H with $O(n^{1+\lambda})$ edges such that $dist_H(u, w) \leq (1 + \epsilon) dist_G(u, w)$ for every pair of vertices $u, w \in V$ such that $dist_G(u, w) \geq \beta(\epsilon, \lambda)$.

We remark that this result is optimal in the sense that the statement becomes false if either ϵ or λ is set to zero. Specifically, taking $\epsilon = 0$ yields a false statement since for any $0 < \lambda < 1$ there exists an infinite graph family $\mathcal{H}(\lambda)$ such that every n-vertex graph $G \in \mathcal{H}(\lambda)$ has $\Omega(n^{1+\lambda})$ edges and for any subgraph H of G there exists a pair of nodes u, w in the graph such that $d_G(u, w) = \Omega(n^{1/2-\lambda/2})$ and $d_H(u, w) > d_G(u, w)$. Analogously, setting λ to zero falsifies the statement because for any $0 < \epsilon < 1$ and $\beta(\epsilon) \geq 1$ there exists an infinite graph family $\hat{\mathcal{H}}(\epsilon, \beta)$ such that any n-vertex graph $G \in \hat{\mathcal{H}}(\epsilon, \beta)$ has $n^{1+\Omega(1/\beta(\epsilon))}$ edges (i.e., significantly more than $O(n) = O(n^{1+\lambda})$)

and for any subgraph H of G there exists a pair of nodes u, w in the graph such that $d_G(u,w) \geq \beta(\epsilon)$ and $d_H(u,w) \geq 2\beta(\epsilon)$ [3].

Furthermore, our construction enables us to obtain a multiplicative stretch that is even smaller than $1+\epsilon$ for constant $\epsilon > 0$, i.e., $1+1/\operatorname{polylog} n$ or even $1+2^{-\log n/\log^{(b)} n}$ (where $\log^{(b)}$ denotes the b-iterated log function), while still keeping the size of the spanner equal to $O(n^{1+\lambda})$ for arbitrarily small $\lambda > 0$, at the cost of increasing the additive term to $\operatorname{polylog} n$ or $\exp(O(\log n/\log^{(b)} n))$, respectively.

A straightforward implementation of our construction requires $O(|E| \cdot n)$ time, but we show that our construction can be implemented even faster, and establish a tradeoff between the time complexity of the construction algorithm and the additive term. Specifically, we show that a $(1+\epsilon, \beta')$-spanner of size $O(\beta' n^{1+1/\kappa})$ can be constructed in $\tilde{O}(n^{2+\mu})$ time for any n-vertex graph G, where the additive term $\beta' = \beta'(\kappa, \epsilon, \mu)$ is constant whenever κ, ϵ, and μ are.

The additive terms involved in our constructions are roughly $\beta(\kappa,\epsilon) = \kappa^{\log\log\kappa - \log\epsilon}$ and $\beta'(\kappa,\epsilon,\mu) = \max\{\beta(\kappa,\epsilon), \kappa^{-\log\mu}\}$. Although optimizing these additive terms is not in the focus of the current paper, we remark that β and β' are always $O((\log n)^{\log^{(3)} n})$, because $\kappa = O(\log n)$. Indeed, for $\kappa = \Omega(\log n)$ the size of the spanner becomes almost linear (i.e., $O(n \cdot (\log n)^{\log^{(3)} n})$).

Finally, analysis of our construction for specific small values of κ enables us to derive some secondary results. First, a construction of an additive 2-spanner of size $O(n^{3/2})$ can be derived. This is tight up to a constant factor due to lower bound of [16], and improves upon the construction of [7] by a logarithmic factor. However, the running time of our construction is $O(n^{5/2})$ instead of $\tilde{O}(n^2)$ of the construction of [7]. Additionally, a construction of a $(1+\epsilon, 4)$-spanner of size $O(\epsilon^{-1} n^{4/3})$ can be derived. This improves the previously known construction of a multiplicative 5-spanner of size $O(n^{4/3})$. The details of the analysis of our algorithm for small values of κ were presented in the preliminary versions of this paper [9, 10] and are omitted here.

Since the appearance of a preliminary version of this paper [10], a more time-efficient construction of $(1+\epsilon, \beta)$-spanners for general graphs was devised in [8]. Specifically, it is shown therein that $(1+\epsilon, \beta'')$-spanners of size $O(\beta'' n^{1+1/\kappa})$ can be constructed in $O(|E| n^{\mu})$ time, where the additive term $\beta'' = \beta''(\kappa, \epsilon, \mu)$ is constant whenever κ, ϵ, and μ are. The improved running time of the construction of [8] makes it possible to use $(1+\epsilon, \beta)$-spanners as a building block of the most efficient currently known algorithm for computing almost shortest paths with constant almost additive error from s, $1 \ll s \ll n$, sources. However, the additive term β'' in the construction of [8] is asymptotically greater than the additive term β' in the present construction.

2. The construction algorithm. In this section we present the polynomial time algorithm that for any constant $0 < \epsilon < 1$ and constant integer $\kappa \geq 2$, and for any n-vertex graph G, constructs a $(1+\epsilon, \beta)$-spanner with $O(\beta n^{1+1/\kappa})$ edges, where $\beta = \beta(\epsilon, \kappa)$ is a constant (independent of n and of any other parameter of the graph).

We start with some necessary definitions. We refer to the vertex and edge sets of a subgraph G' by $V(G')$ and $E(G')$, respectively. For any subgraph $G' = (V', E')$ and any integer $\ell \geq 0$ and vertex $v \in V'$, we denote the ℓ-neighborhood of v in G' by $\Gamma_\ell^{G'}(v) = \{u \mid dist_{G'}(v,u) \leq \ell\}$. For any subset of vertices $U \subseteq V'$, denote the set of neighbors of U in G' by $\Gamma^{G'}(U) = \{z \mid \exists u \in U \text{ such that } (u,z) \in E'\}$. We also denote $\Gamma_\ell(v) = \Gamma_\ell^G(v)$ and $\Gamma(U) = \Gamma^G(U)$.

Input: Graph $G=(V,E)$.
Output: Spanned partition $\mathcal{G}$, subgraph H.

1. $U \leftarrow V$; $\mathcal{G} \leftarrow \emptyset$; $\mathcal{S}' \leftarrow \emptyset$; $E(H) \leftarrow \emptyset$;
2. **While** $U \neq \emptyset$ **do:**
   (a) Pick an arbitrary vertex $v \in U$;
   (b) $S \leftarrow \{v\}$;
   (c) **While** $|S \cup \Gamma(S) \cap U| \geq n^{1/\kappa}|S|$ **do:**
   - $S \leftarrow S \cup \Gamma(S) \cap U$; /* the cluster
   (d) $\hat{S} \leftarrow S \cup \Gamma(S) \cap U$; /* the shell
   (e) Form a BFS spanning tree T for (v,S);
   (f) $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{(v,\hat{S})\}$; $\mathcal{G} \leftarrow \mathcal{G} \cup \{(v,S,T)\}$; $U \leftarrow U \setminus S$;
3. **For every** $(v,\hat{S}) \in \mathcal{S}'$ **do:**
   (a) Create a BFS spanning tree T' rooted at v for $\hat{S}$ (namely, a tree spanning $\hat{S}$ and yielding shortest paths to v in the induced subgraph $G(\hat{S})$);
   (b) $E(H) \leftarrow E(H) \cup E(T')$;
4. Return$(\mathcal{G},H)$;

FIG. 1. *Procedure* DOWN_PART.

For a subset of vertices $W \subseteq V$, let $G(W)$ denote the subgraph of G induced by W. With a slight abuse of notation, for a subset $E' \subseteq E$ of edges, let $G(E')$ denote the graph $(V(E'),E')$, where $V(E') = \{v \in V \mid \exists e \in E' s.t.\ v \in e\}$. A subset of vertices $C \subseteq V$ is called a *cluster* if its induced subgraph $G(C)$ is connected. A triple (v,S,T) is called a *spanned cluster* if $v \in S$, and T is a connected spanning tree of S. (Note that S itself is not necessarily connected, and the tree T may span also some vertices that do not belong to S.) The *radius* of a spanned cluster (v,S,T) is $rad(v,S,T) = \max\{dist_{G(T)}(v,u) \mid u \in S\}$, and the *diameter* is $diam(v,S,T) = \max\{dist_{G(T)}(u,w) \mid u,w \in S\}$. Since in a spanned cluster (v,S,T), the tree T is necessarily connected, the radius and the diameter of a spanned cluster are always finite. The spanned clusters (v_i,S_i,T_i) and (v_j,S_j,T_j) are *disjoint* if $S_i \cap S_j = \emptyset$. A set of disjoint spanned clusters $\{(v_i,S_i,T_i)\}$ is called a *spanned partition*. For any pair of vertex sets $U_1,U_2 \subseteq V$ denote $dist_G(U_1,U_2) = \min\{dist_G(u_1,u_2) \mid u_1 \in U_1, u_2 \in U_2\}$. For a spanned partition $\mathcal{U}$, a cluster S, and an integer ℓ, denote

$$\Gamma^{\mathcal{U}}_{\ell}(S) \;=\; \{(v_i,S_i,T_i) \in \mathcal{U} \mid dist_G(S_i,S) \leq \ell\}\ .$$

2.1. The initial partitioning procedure. Our algorithm starts by invoking Procedure DOWN_PART, described in Figure 1. This procedure is a variant of the procedures for constructing partitions due to [14, 7]. The procedure creates a spanned partition $\mathcal{G}$, which we also call the *ground partition*, since the algorithm uses it as a basis for constructing other partitions. (This partition satisfies that $\bigcup_i S_i = V$; the partitions constructed later on may be *partial*, i.e., they will not necessarily have this property.) For each cluster that the procedure creates, it also builds a "shell" consisting of the cluster and one external layer, in a way similar to the partitioning algorithm of [2]. The procedure also creates a subgraph H, which is a subset, and in some sense a *core*, of the spanner created by the algorithm. This subgraph H consists of the union of the breadth first search (BFS) trees of all the shells created by the procedure. (Given a cluster S centered at a vertex v in the graph G, a *BFS tree* for (v,S) is a tree spanning S which yields shortest paths to v in the induced subgraph $G(S)$.)

Let us next establish some basic properties concerning the output of Procedure Down_Part. Given a spanned partition $\mathcal{S} = \{(v_j, S_j, T_j)\}$, denote by $\mathcal{A}_i(\mathcal{S})$ the subset of spanned clusters of $\mathcal{S}$ with radius i,

$$\mathcal{A}_i(\mathcal{S}) \;=\; \{(v,S,T) \in \mathcal{S} \mid rad(v,S,T) = i\}.$$

For any spanned partition $\mathcal{S}$ denote the minimum cluster size by $\check{S}(\mathcal{S}) = \min_{S\in\mathcal{S}} |S|$.

Lemma 2.1. *Let $\mathcal{G}$ be the ground partition returned by Procedure* Down_Part. *Then $\check{S}(\mathcal{A}_j(\mathcal{G})) \geq n^{j/\kappa}$ for any integer $0 \leq j \leq \kappa - 1$.*

Proof. Consider a spanned cluster $(v,S,T) \in \mathcal{A}_j(\mathcal{G})$. By definition of $\mathcal{A}_j(\mathcal{G})$, $rad(v,S,T) = j$. Each cluster constructed by Procedure Down_Part starts with radius zero, and each iteration of the internal while loop (step 2(c)) of the procedure increases the radius of the cluster by at most 1. Hence the cluster (v,S,T) was involved in the loop for at least j iterations. In each iteration its size grew by a factor of at least $n^{1/\kappa}$. The lemma follows.

Next, we argue that the subgraph H returned by Procedure Down_Part is sparse.

Lemma 2.2. $|E(H)| \;=\; O(n^{1+1/\kappa})$.

Proof. For every cluster $S \in \mathcal{S}$, the shell consists of $\hat{S} = S \cup \Gamma(S) \cap U$ at the time of insertion, so by the condition of step 2(c) for selecting the cluster, $|\hat{S}| < n^{1/\kappa} \cdot |S|$. Hence the number of edges inserted into H in step 3 of Procedure Down_Part is bounded by

$$\sum_{S\in\mathcal{S}} |S| n^{1/\kappa} \;=\; n^{1/\kappa} \sum_{S\in\mathcal{S}} |S| \;=\; O(n^{1+1/\kappa}),$$

since the clusters S are disjoint. ∎

An additional important property of the ground partition $\mathcal{G}$ and the subgraph H returned by the invocation Down_Part(G) is that for any pair of neighboring clusters of $\mathcal{G}$ and for any edge e between these clusters, there is an edge between these clusters in H that is incident to one of the endpoints of e. More specifically, we introduce the following key definition.

Definition 2.3. *A spanned partition $\mathcal{S}$ of a graph $G = (V,E)$ is said to be* adjacency-preserving *with respect to a subgraph H such that $E(H) \subseteq E$ if it satisfies the following two properties.*

(P1) *For any spanned cluster $(v,S,T) \in \mathcal{S}$, there exists a BFS spanning tree T' for (v,S) such that $E(T') \subseteq E(H)$.*

(P2) *For any pair of neighboring clusters $(v_1,S_1,T_1), (v_2,S_2,T_2) \in \mathcal{S}$ (i.e., such that $dist_G(S_1,S_2) = 1$) and for any edge $e = (u_1,u_2) \in E$ such that $u_1 \in S_1$ and $u_2 \in S_2$, there exists either a node $u_2' \in S_2$ such that $(u_1,u_2') \in E(H)$ or a node $u_1' \in S_1$ such that $(u_1',u_2) \in E(H)$. In the former (resp., latter) case, the edge e is said to be* spanned through *the cluster S_2 (resp., S_1).*

Note that in the definition above, if the edge (u_1,u_2) is spanned through the cluster S_i, for $i \in \{1,2\}$, then $dist_H(u_1,u_2) \leq diam(S_i) + 1$.

Lemma 2.4. *Let $(\mathcal{G},H)$ be the pair returned by Procedure* Down_Part(G). *Then the spanned partition $\mathcal{G}$ is adjacency-preserving with respect to H.*

Proof. To prove that the pair $(\mathcal{G},H)$ satisfies property (P1) of Definition 2.3, consider some spanned cluster (v,S,T). Note that at step 3(a) of Procedure Down_Part, the BFS spanning tree T' of its shell $\hat{S}$ rooted at v was inserted into the subgraph H. Note that such a tree is, in particular, a BFS spanning tree for S rooted at v (although not necessarily the same as T), completing the proof of property (P1).

To prove that $(\mathcal{G}, H)$ satisfies property (P2) as well, consider some pair of neighboring spanned clusters (v_1, S_1, T_1) and (v_2, S_2, T_2), and some edge $e = (u_1, u_2)$ between them, such that $u_1 \in S_1$ and $u_2 \in S_2$. Assume, without loss of generality, that the cluster S_1 was created before the cluster S_2. Consider the iteration of the main loop of Procedure Down_Part on which S_1 was created. Denote by U' the set U at the beginning of this iteration. Note that at this stage all the nodes of S_1 and S_2 were still uncovered, i.e., $S_1, S_2 \subseteq U'$, and thus in particular $u_1, u_2 \in U'$. It follows that e is in $G' = G(U')$, the subgraph of G induced by U', and hence $u_2 \in \Gamma^{G'}(S_1) \subseteq \hat{S}_1$.

Denote the radius of S_1 by $\rho = rad(v_1, S_1, T_1)$. By step 2(c) of Procedure Down_Part, $\Gamma^{G'}_{\rho}(v_1) = S_1$ and $\Gamma^{G'}_{\rho+1}(v_1) = \hat{S}_1$. Recalling that $u_2 \in \hat{S}_1 \setminus S_1$, we conclude that $dist_{G'}(v_1, u_2) = \rho + 1$.

The spanning tree T' constructed for $\hat{S}_1$ at step 3(a) of Procedure Down_Part spans $\hat{S}_1$, and thus $u_2 \in V(T')$. Since T' is a BFS spanning tree with respect to v_1 in the induced subgraph $G'(\hat{S}_1)$, and since $dist_{G'}(v_1, u_2) = \rho + 1$, it follows that the parent z_1 of u_2 in T' satisfies $dist_{G'}(v_1, z_1) = \rho$, and thus $z_1 \in \Gamma^{G'}_{\rho}(v_1) = S_1$, as required. ∎

2.2. The superclustering procedure. We next proceed to describing the *superclustering* procedure SC. The procedure receives a spanned partition $\mathcal{C}$, identifies its "dense" sets of clusters, and merges them into *superclusters*. By a dense set of clusters we mean a set containing "fairly many" clusters, which are "close enough" to each other. These qualitative notions are quantified by two additional parameters. The *size* parameter σ of Procedure SC specifies the minimum number of clusters close to a given cluster S that justifies creating a supercluster around S that will contain them, whereas the *distance* parameter δ specifies when two clusters are considered to be close. Procedure SC returns a spanned partition $\mathcal{C}'$ which contains the superclusters created and a subgraph H that will later be added into the spanner. When Procedure SC finishes creating superclusters, it remains with a collection $\mathcal{R}$ of original clusters of the input partition that were left untouched. Procedure SC then finds shortest paths between close pairs in the collection $\mathcal{R}$ and inserts these paths as well into the output subgraph H. The size parameter σ controls also the number of pairs of close $\mathcal{R}$ clusters and therefore the size of the subgraph H. The procedure is described in Figure 2.

Note that in step 2(b)ii, while augmenting the set of edges T' associated with a supercluster S' by adding the edges of the shortest path P_i, we do not insert the vertices of this path into the vertex set S' associated with this supercluster. The reason is that these vertices are clustered in other clusters of the ground partition and they may be superclustered separately. In other words, the same vertex may participate in the shortest path connecting two clusters of one supercluster, and at the same time be clustered in another supercluster. Hence T' is not necessarily contained in $E(G(S'))$.

Intuitively, the following lemma states that Procedure SC does not destroy the adjacency-preservation property.

Lemma 2.5. *Let the triple $(\mathcal{C}', H', \mathcal{R})$ be the output of the invocation* SC$(G, \mathcal{C}, \sigma, \delta)$ *for a graph $G = (V, E)$, its spanned partition $\mathcal{C}$ and some arbitrary σ and δ. Suppose that the spanned partition $\mathcal{C}$ is adjacency-preserving with respect to some subgraph H of G such that $E(H) \subseteq E$. Then the output spanned partitions $\mathcal{C}'$ and $\mathcal{R}$ are adjacency-preserving with respect to $H \cup H'$.*

Proof. Note first that the adjacency-preservation property is monotone in both parameters (i.e., if a spanned partition $\tilde{\mathcal{S}}$ is adjacency-preserving with respect to

Input: Graph $G = (V, E)$, spanned partition $\mathcal{C} = \{(v_i, S_i, T_i)\}$ of G, integers σ, $\delta \geq 1$.
Output: Spanned partition $\mathcal{C}' = \{(v'_i, S'_i, T'_i)\}$ of G, subgraph H', collection $\mathcal{R}$ of unmerged clusters.

1. $E(H') \leftarrow \emptyset$; $\mathcal{U} \leftarrow \mathcal{C}$; $\mathcal{C}' \leftarrow \emptyset$;
2. **while** there exists a spanned cluster $(v, S, T) \in \mathcal{U}$ such that $|\Gamma^{\mathcal{U}}_{\delta}(S) \cap \mathcal{U}| \geq \sigma$ **do:**
   (a) $S' \leftarrow \bigcup_{(v_i, S_i, T_i) \in \Gamma^{\mathcal{U}}_{\delta}(S)} S_i$;
   (b) i $T' \leftarrow T$;
   ii **For** every S_i such that $(v_i, S_i, T_i) \in \Gamma_{\mathcal{U}}(S)$ **do:**
   A. Compute the shortest path P_i in G between the clusters S and S_i;
   B. $E(T') \leftarrow E(T') \cup E(P_i) \cup E(T_i)$;
   (c) $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{(v, S', T')\}$;
   (d) $\mathcal{U} \leftarrow \mathcal{U} \setminus \Gamma^{\mathcal{U}}_{\delta}(S)$;
   (e) $E(H') \leftarrow E(H') \cup E(T')$;
3. $\mathcal{R} \leftarrow \mathcal{U}$; /* Remaining (unmerged) clusters */
4. **For** every pair of spanned clusters $(v_i, S_i, T_i), (v_j, S_j, T_j) \in \mathcal{R}$ such that $dist_G(S_i, S_j) \leq \delta$ **do:**
   (a) Compute the shortest path P_{ij} between S_i and S_j in G;
   (b) $E(H') \leftarrow E(H') \cup E(P_{ij})$;
5. Return$(\mathcal{C}', H', \mathcal{R})$;

FIG. 2. *Procedure* SC.

some subgraph $\tilde{H}$, then this spanned partition is adjacency-preserving with respect to any subgraph $\bar{H}$ such that $E(\tilde{H}) \subseteq E(\bar{H}) \subseteq E$). Also, if $\bar{\mathcal{C}} \subseteq \tilde{\mathcal{C}}$, and $\tilde{\mathcal{C}}$ is adjacency-preserving with respect to some subgraph $\tilde{H}$, then the spanned partition $\bar{\mathcal{C}}$ is adjacency-preserving with respect to $\tilde{H}$ as well.

Now the statement of the lemma regarding the spanned partition $\mathcal{R}$ follows from the observation that $\mathcal{R} \subseteq \mathcal{C}$, and from the assumption of the lemma that $\mathcal{C}$ is adjacency-preserving with respect to H.

It remains to establish the lemma for $\mathcal{C}'$. Observe that the pair $(\mathcal{C}', H \cup H')$ satisfies property (P1) of Definition 2.3, by the same assumption of the lemma, and by step 2(e) of Procedure SC. To prove that the pair $(\mathcal{C}', H \cup H')$ satisfies property (P2) as well, consider a pair of neighboring superclusters $(v_1, S_1, T_1), (v_2, S_2, T_2) \in \mathcal{C}'$, and an edge $(u_1, u_2) \in E$ between them such that $u_1 \in S_1$ and $u_2 \in S_2$. By step 2(a) of Procedure SC, $S_1, S_2 \subseteq \bigcup_{\tilde{S}_i \in \mathcal{C}} \tilde{S}_i$. Hence there exist clusters $\tilde{S}_i, \tilde{S}_j \in \mathcal{C}$ such that (a) $\tilde{S}_i \subseteq S_1$, (b) $\tilde{S}_j \subseteq S_2$, (c) $u_1 \in \tilde{S}_i$, and (d) $u_2 \in \tilde{S}_j$. Since $\mathcal{C}$ is adjacency-preserving with respect to H, it follows from (c) and (d) that there exists either a node $u_j \in \tilde{S}_j$ such that the edge (u_1, u_j) is in $E(H)$, or a node $u_i \in \tilde{S}_i$ such that the edge (u_i, u_2) is in $E(H)$. In either case, we are done by (a) and (b). □

2.3. The main algorithm. We proceed with the description of our main algorithm, named Algorithm SP_CONS. Our construction uses parameters κ, J, and Υ, to be fixed explicitly later on. The following analysis is valid for any nonnegative integers κ, J, and Υ that satisfy $1 \leq J \leq \lceil \log \kappa \rceil$ and $\Upsilon = \Omega((\kappa/2^{J-3})^J)$. Set $t_j = (\kappa - 2^{j-1}) \,/\, (2^{j-1}\kappa)$ and $\delta_j = \delta_j(\Upsilon, J) = \Upsilon^{j/J}$ for every $1 \leq j \leq J$. Also for every $1 \leq j \leq J-1$ set $\sigma_j = n^{t_{J-j} - t_{J-j+1}} = n^{2^{j-J}}$. Denote $\tau_j = [\kappa t_{J+1-j}, \kappa t_{J-j})$ for

Input: Graph $G=(V,E)$, integers $\kappa, J, \Upsilon > 1$.
Output: Subgraph H

1. $(\mathcal{G}, H) \leftarrow$ DOWN_PART(G);
2. $\mathcal{C}' \leftarrow \emptyset$;
3. **For** $j = 1$ to $J-1$ **do:**
   (a) $\mathcal{C} \leftarrow \mathcal{C}' \cup \bigcup_{i\in\tau_j} \mathcal{A}_i(\mathcal{G})$;
   (b) $(\mathcal{C}', H', \mathcal{R}) \leftarrow \mathrm{SC}(G, \mathcal{C}, \sigma_j, 2\delta_j(\Upsilon, J))$;
   (c) $E(H) \leftarrow E(H) \cup E(H')$;
4. $\mathcal{R} \leftarrow \mathcal{C}' \cup \bigcup_{i\in\tau_J} \mathcal{A}_i(\mathcal{G})$;
5. **For** every pair of spanned clusters (v_i, S_i, T_i), $(v_j, S_j, T_j) \in \mathcal{R}$ such that $dist_G(S_i, S_j) \le 2\delta_J(\Upsilon, J)$ **do:**
   (a) Calculate the shortest path P_{ij} between S_i and S_j;
   (b) $E(H) \leftarrow E(H) \cup E(P_{ij})$.
6. Return H as the resulting spanner.

FIG. 3. *Algorithm* SP_CONS.

$1 \le j \le J-1$ and $\tau_J = [\kappa t_1, \kappa)$. Observe that for $j \le J \le \lceil \log\kappa \rceil$, t_j is nonnegative. The parameters Υ and J will be chosen in a such a way that $\Upsilon^{1/J}$ will be a rather large constant, so the parameters δ_j for $1 \le j \le J$ are just the consecutive powers of this constant. These powers will serve as distance thresholds, and will grow as the algorithm proceeds.

As already mentioned, the algorithm starts by forming the ground partition $\mathcal{G}$. The ground partition contains a subset $\mathcal{Z} = \bigcup_{i<t_J\kappa} \mathcal{A}_i(\mathcal{G})$ of *singleton* clusters, that is, clusters that contain single vertex, and therefore have radius 0 (these properties of the clusters of the set $\mathcal{Z}$ will be ensured by the appropriate choice of the parameters J and κ). After forming the ground partition $\mathcal{G}$, the algorithm invokes iteratively Procedure SC. On each iteration the procedure forms a new spanned partition with fewer clusters (each of which is larger), and also takes care of clusters that were not merged into superclusters, by interconnecting pairs of nearby clusters by paths which are added to the subgraph H. More specifically, the input spanned partition $\mathcal{C}$ of Procedure SC on iteration j, $j = 1, 2, \ldots, J-1$, is the union of the output spanned partition $\mathcal{C}'$ of the previous $(j-1)$st iteration (for $j = 0$, $\mathcal{C}' = \emptyset$), and of the appropriate subset $Q_j = \bigcup_{i\in\tau_j} \mathcal{A}_i(\mathcal{G})$. The radii and the sizes of the clusters of Q_j grow with j. Algorithm SP_CONS repeats these iterations until we are left with a sufficiently small number of large clusters. The pairs of nearby clusters among these large clusters in the final spanned partition are again interconnected by shortest paths. These shortest paths are inserted into the subgraph H. At the end of this process, H contains the spanning trees of the shells of all the clusters and superclusters created through the process, and also all the shortest paths between clusters and superclusters that were created throughout the execution. Note that the notion of a "nearby" pair of clusters changes from one iteration of the algorithm to another. Specifically, on iteration j, pairs of clusters at distance $\delta_j(\Upsilon, \kappa)$ are considered close. This change corresponds to the increase of clusters radii.

A formal description of the algorithm is given in Figure 3. Figure 4 provides a schematic illustration.

Remark. The spanned partition $\mathcal{R}$ returned by Procedure SC is not used by the main algorithm; it is output by the procedure for the convenience of the analysis only.

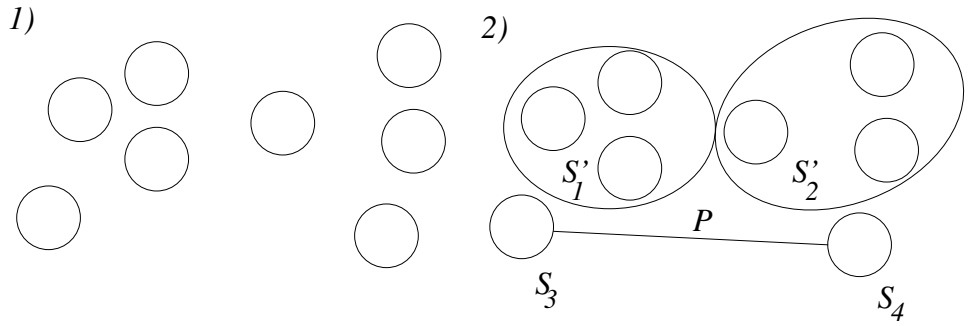


FIG. 4. 1) *The ground partition $\mathcal{G}$ is obtained by applying Procedure* DOWN_PART *to the graph G.* 2) *The two superclusters S'_1 and S'_2 were formed by an application of Procedure* SC. *The two clusters S_3 and S_4 were not merged into a supercluster. Thus, the shortest path P between them was inserted into the subgraph H.*

3. Analysis. To begin with, it is not hard to see that Algorithm SP_CONS runs in polynomial time. Indeed, each invocation of Procedure SC requires essentially at most n BFS explorations of the graph, and the same is true regarding Procedure DOWN_PART. The precise analysis of its running time is deferred to section 4.

We proceed with an analysis of the properties of the algorithm.

3.1. Bounding the cluster diameters. We first bound the diameters of the clusters and superclusters that are created in different iterations of Algorithm SP_CONS. For a spanned partition $\mathcal{C}$ denote $\hat{D}(\mathcal{C}) = \max_{(v,S,T)\in\mathcal{C}}\{diam(v,S,T)\}$ (for an empty spanned partition $\mathcal{C} = \emptyset$, let $\hat{D}(\mathcal{C}) = 0$). Since the graph G remains the same in all the invocations of Procedure SC, we henceforth omit it from its list of parameters.

LEMMA 3.1. *Let $\mathcal{C}'$ be spanned partition output by the invocation* $\mathrm{SC}(\mathcal{C},\sigma,\delta)$ *of Procedure* SC. *Then $\hat{D}(\mathcal{C}') \le 3\hat{D}(\mathcal{C}) + 2\delta$.*

Proof. Consider some supercluster $(v', S', T') \in \mathcal{C}'$. It was created at step 2(b)ii of Procedure SC. Therefore, it has the form of a star, with a cluster $(v' = v_0, S_0, T_0) \in \mathcal{C}$ in the middle, connected to some $b \ge \sigma$ clusters $(v_1, S_1, T_1), \ldots, (v_b, S_b, T_b) \in \mathcal{C}$ by shortest paths $P_1, \ldots, P_b$ of length at most δ. Consider a pair of vertices $z_i \in S_i$, $z_j \in S_j$ such that $1 \le i < j \le b$. Let $u_i \in S_i$ and $w_i \in S_0$ be the endpoints of the path P_i, and let $u_j \in S_j$ and $w_j \in S_0$ be the endpoints of the path P_j. Then

$$\begin{aligned} dist_{G(T')}(z_i,z_j) \le\ & dist_{G(T')}(z_i,u_i) + dist_{G(T')}(u_i,w_i) + dist_{G(T')}(w_i,w_j) \\ & + dist_{G(T')}(w_j,u_j) + dist_{G(T')}(u_j,z_j)\ . \end{aligned}$$

As by step 2(b)iiB of Procedure SC, $E(T_0), E(T_i), E(T_j) \subseteq E(T')$, and $z_i, u_i \in V(T_i)$, $w_i, w_j \in V(T_0)$ and $u_j, z_j \in V(T_j)$, it follows that

$$\begin{aligned} dist_{G(T')}(z_i,u_i) &\le dist_{G(T_i)}(z_i,u_i) \ \le\ diam(v_i,S_i,T_i)\ , \\ dist_{G(T')}(w_i,w_j) &\le dist_{G(T_0)}(w_i,w_j) \ \le\ diam(v_0,S_0,T_0)\ , \\ dist_{G(T')}(u_j,z_j) &\le dist_{G(T_j)}(u_j,z_j) \ \le\ diam(v_j,S_j,T_j)\ . \end{aligned}$$

Note that $diam(v_i,S_i,T_i), diam(v_0,S_0,T_0), diam(v_j,S_j,T_j) \le \hat{D}(\mathcal{C})$. Also, (by step 2(b)iiB of Procedure SC), $E(P_i), E(P_j) \subseteq E(T')$, and $u_i, w_i \in V(P_i)$, $u_j, w_j \in V(P_j)$,

implying that

$$dist_{G(T')}(u_i,w_i) \le dist_{G(P_i)}(u_i,w_i) \le \delta\ ,$$
$$dist_{G(T')}(u_j,w_j) \le dist_{G(P_j)}(u_j,w_j) \le \delta\ .$$

To conclude

$$\begin{aligned} dist_{G(T')}(z_i,z_j) &\le diam(v_i,S_i,T_i) + \delta + diam(v_0,S_0,T_0) + \delta + diam(v_j,S_j,T_j) \\ &\le 3\hat{D}(\mathcal{C}) + 2\delta. \end{aligned}$$

It is easy to see that this bound applies to the distance between any pair of vertices z_i, z_j in S'. □

For any $1 \le j \le J-1$, let $\mathcal{C}_j$ be the input partition of the jth invocation of Procedure SC and let $(\mathcal{C}'_j, H_j, \mathcal{R}_j)$ be the output returned by this invocation. The collection of clusters $\mathcal{R}_J$ is defined to be the set $\mathcal{R}$ created in step 4 of Algorithm Sp_Cons. We refer to clusters of $\mathcal{R}_j$ as the *jth level clusters* of the resulting partition. Note that these clusters are never merged again in subsequent iterations.

By steps 1 and 3 of Procedure SC, and since after step 1 of Procedure SC no cluster is inserted into the collection of uncovered spanners $\mathcal{U}$, we have that for every $1 \le j \le J-1$,

$$\mathcal{R}_j \ \subseteq\ \mathcal{C}_j. \tag{1}$$

Also, for every $1 \le j \le J-1$, since on the jth iteration of Algorithm Sp_Cons, Procedure SC is invoked with distance parameter δ_j, by Lemma 3.1

$$\hat{D}(\mathcal{C}'_j) \ \le\ 3\hat{D}(\mathcal{C}_j) + 2\delta_j. \tag{2}$$

Lemma 3.2. *For every integer $0 \le j \le J-2$,*

$$\mathcal{R}_{j+1} \subseteq \mathcal{C}_{j+1} \ =\ \mathcal{C}'_j \cup \bigcup_{i\in\tau_{j+1}} \mathcal{A}_i(\mathcal{G}).$$

Also, $\mathcal{R}_J \subseteq \mathcal{C}'_{J-1} \cup \bigcup_{i\in\tau_J} \mathcal{A}_i(\mathcal{G})$.

Proof. The first statement of the lemma follows by step 3(a) of Algorithm Sp_Cons and using (1). The second statement follows from the definition of $\mathcal{R}_J$ and step 4 of Algorithm Sp_Cons. □

Lemma 3.3. *For every integer $1 \le j \le J-1$,*

$$\text{(a)}\quad \hat{D}(\mathcal{C}'_j) \le 2\left(3^j\kappa t_{J-1} + \sum_{l=1}^{j} 3^{j-l}\Upsilon^{l/J}\right),$$
$$\text{(b)}\quad \hat{D}(\mathcal{C}_j) \le 2\left(3^{j-1}\kappa t_{J-1} + \sum_{l=1}^{j-1} 3^{j-1-l}\Upsilon^{l/J}\right).$$

Proof. By induction on j. For the induction base for $j=1$, note that the first invocation of Procedure SC is with $\mathcal{C} = \bigcup_{i\in\tau_1} \mathcal{A}_i(\mathcal{G})$. By definition of $\mathcal{A}_i(\mathcal{G})$ we have $\hat{D}(\mathcal{C}) < \hat{D}(\mathcal{A}_{t_{J-1}\kappa}(\mathcal{G})) \le 2t_{J-1}\kappa$, establishing claim (b). Claim (a) now follows by inequality (2).

For the induction step, assume the claims for j between 1 and $J-2$ and consider $j+1$. The parameter δ in the invocation of Procedure SC that formed the clusters of $\mathcal{C}'_{j+1}$ was equal to $\delta_{j+1} = \Upsilon^{(j+1)/J}$. By the induction hypothesis

$$\hat{D}(\mathcal{C}'_j) \;\leq\; 2\left(3^j \kappa t_{J-1} + \sum_{l=1}^{j} 3^{j-l}\Upsilon^{l/J}\right). \tag{3}$$

By step 3(a) of Algorithm Sp_Cons, $\mathcal{C}_{j+1} = \mathcal{C}'_j \cup \bigcup_{i\in\tau_{j+1}} \mathcal{A}_i(\mathcal{G})$. Hence

$$\hat{D}(\mathcal{C}_{j+1}) \;\leq\; \max\left\{\hat{D}(\mathcal{C}'_j), \hat{D}\left(\bigcup_{i\in\tau_{j+1}} \mathcal{A}_i(\mathcal{G})\right)\right\}. \tag{4}$$

By definition of $\mathcal{A}_i(\mathcal{G})$ and t_j

$$\hat{D}(\bigcup_{i\in\tau_{j+1}} \mathcal{A}_i(\mathcal{G})) \;\leq\; 2\kappa t_{J-(j+1)} \;\leq\; \frac{\kappa}{2^{J-j-3}}\,.$$

Recall that $\Upsilon^{1/J} = \Omega(\kappa/2^{J-3})$. Also, if Υ is set in such a way that $\Upsilon^{1/J}$ is a sufficiently large constant (e.g., $\Upsilon^{1/J} \geq 4$), then $2^j \leq \Upsilon^{(j-1)/J}$, and thus

$$\hat{D}\left(\bigcup_{i\in\tau_{j+1}} \mathcal{A}_i(\mathcal{G})\right) \;\leq\; \frac{\kappa}{2^{J-j-3}} \;\leq\; \Upsilon^{j/J} \;\leq\; \sum_{l=1}^{j} 3^{j-l}\Upsilon^{l/J}\,. \tag{5}$$

Now using inequalities (3), (4), and (5), we get

$$\hat{D}(\mathcal{C}_{j+1}) \;\leq\; 2\left(3^j \kappa t_{J-1} + \sum_{l=1}^{j} 3^{j-l}\Upsilon^{l/J}\right),$$

yielding claim (b). Also by inequality (2),

$$\begin{aligned}\hat{D}(\mathcal{C}'_{j+1}) \;&\leq\; 3\cdot 2\left(3^j \kappa t_{J-1} + \sum_{l=1}^{j} 3^{j-l}\Upsilon^{l/J}\right) + 2\Upsilon^{(j+1)/J}\\ &=\; 2\left(3^{j+1} \kappa t_{J-1} + \sum_{l=1}^{j+1} 3^{j+1-l}\Upsilon^{l/J}\right),\end{aligned}$$

yielding claim (a). □

Corollary 3.4. *For every integer $1 \leq j \leq J$,*

$$\hat{D}(\mathcal{R}_j) \;\leq\; 2\left(3^{j-1} \kappa t_{J-1} + \sum_{l=1}^{j-1} 3^{j-1-l}\Upsilon^{l/J}\right).$$

Proof. For $1 \leq j \leq J-1$ the claim follows from Lemma 3.3 by (1). For $j = J$, we get by step 4 of Algorithm Sp_Cons that

$$\hat{D}(\mathcal{R}_J) \;\leq\; \max\{\kappa, \hat{D}(\mathcal{C}'_{J-1})\} \;\leq\; 2\left(3^{J-1} \kappa t_{J-1} + \sum_{l=1}^{J-1} 3^{J-l}\Upsilon^{l/J}\right). \tag{6}$$

□

3.2. Bounding the construction size. In this section we bound the size of the subgraph returned by Algorithm Sp_Cons. First, by Lemma 2.2, the number of edges inserted into the subgraph H by Procedure Down_Part (i.e., by step 1 of Algorithm Sp_Cons) is $O(n^{1+1/\kappa})$. Next, we analyze the number of edges inserted into the subgraph H at step 3 of Algorithm Sp_Cons.

As each of the superclusters created by Procedure SC contains at least σ (disjoint) clusters of the input partition $\mathcal{C}$, we have the following.

LEMMA 3.5. *Let $\mathcal{C}'$ denote the spanned partition output by an invocation* $\mathrm{SC}(\mathcal{C},\sigma,\delta)$ *of Procedure* SC. *Then $\check{S}(\mathcal{C}') \geq \sigma \cdot \check{S}(\mathcal{C})$.*

LEMMA 3.6. *Let H denote the subgraph output by an invocation of Procedure* $\mathrm{SC}(\mathcal{C},\sigma,\delta)$. *Then $|E(H)| = O(n\sigma\delta/\check{S}(\mathcal{C}))$.*

Proof. Note that by definition of spanned partition, $\mathcal{C}$ is a collection of disjoint clusters. Hence $|\mathcal{C}| \leq n/\check{S}(\mathcal{C})$.

Consider the supergraph G' in which every vertex represents a cluster of $\mathcal{C}$ and there is an edge between two vertices if and only if the corresponding two clusters are at distance of at most δ in G. Step 2 of Procedure SC creates a forest $\mathcal{F}$ of disjoint star-like trees covering a subset of vertices of G'. It then inserts into H the spanning trees of all the clusters of $\mathcal{C}$ corresponding to the vertices of G' covered by $\mathcal{F}$ (summing up to $O(n)$ edges) and the paths in G corresponding to the edges of $\mathcal{F}$. As $\mathcal{F}$ is a forest,

$$|E(\mathcal{F})| \;\leq\; |V(\mathcal{F})| \;\leq\; |\mathcal{C}| \;\leq\; n/\check{S}(\mathcal{C})\;.$$

Also each path represented by an edge of $\mathcal{F}$ has length of at most δ. Hence step 2 of Procedure SC inserts at most $n\delta/\check{S}(\mathcal{C})$ edges into H.

Note that after removing from G' the vertices covered by $\mathcal{F}$, the removed supergraph has maximal degree of at most σ. Hence the number of remaining edges is at most $|\mathcal{C}|\sigma \leq n\sigma/\check{S}(\mathcal{C})$. Step 4 inserts into H all the paths corresponding to the edges left in G'. Hence it inserts at most $n\sigma\delta/\check{S}(\mathcal{C})$ edges.

It follows that overall, the number of edges inserted by Procedure SC is at most

$$n\sigma\delta/\check{S}(\mathcal{C}) + n\delta/\check{S}(\mathcal{C}) + n \;=\; O(n\sigma\delta/\check{S}(\mathcal{C})). \qquad \square$$

LEMMA 3.7. *For any $1 \leq j \leq J-1$, $\check{S}(\mathcal{C}_j) \geq n^{t_{J-j+1}}$.*

Proof. The proof is by induction on j. The induction base is $j=1$. Recall that $\mathcal{G}$ is the ground partition returned by Procedure Down_Part. In iteration 1,

$$\check{S}(\mathcal{C}_1) \;=\; \check{S}\left(\bigcup_{i\in\tau_1}\mathcal{A}_i(\mathcal{G})\right) \;=\; \check{S}(\mathcal{A}_{t_J\kappa}(\mathcal{G})) \geq n^{t_J}$$

by Lemma 2.1.

For the induction step, let $1 < j < J-1$, and assume (by the inductive hypothesis) that $\check{S}(\mathcal{C}_j) \geq n^{t_{J-j+1}}$. Since $\sigma_j = n^{t_{J-j}-t_{J+1-j}}$, Lemma 3.5 implies that

$$\check{S}(\mathcal{C}'_j) \;\geq\; \check{S}(\mathcal{C})\cdot\sigma_j \geq n^{t_{J+1-j}}\cdot n^{t_{J-j}-t_{J+1-j}} = n^{t_{J-j}}.$$

Also $\mathcal{C}_{j+1} = \mathcal{C}_j \cup \bigcup_{i\in\tau_{j+1}}\mathcal{A}_i(\mathcal{G})$. Hence $\check{S}(\mathcal{C}_{j+1}) \geq \min\{\check{S}(\mathcal{C}'_j), \check{S}(\bigcup_{i\in\tau_{j+1}}\mathcal{A}_i(\mathcal{G}))\}$. Since by Lemma 2.1, $\check{S}(\bigcup_{i\in\tau_{j+1}}\mathcal{A}_i(\mathcal{G})) \geq n^{t_{J-j}}$, the lemma follows. $\square$

LEMMA 3.8. *At the end of step 3 of Algorithm* Sp_Cons, *$|E(H)| = O(\Upsilon n^{1+1/\kappa})$.*

Proof. By Lemma 3.7, $|\mathcal{C}_j| \leq n/\check{S}(\mathcal{C}_j) \leq n^{1-t_{J-j+1}}$. Hence by Lemma 3.6, in the jth iteration, Procedure SC inserts into the output subgraph H_j no more than $\delta_j n^{1-t_{J-j+1}}\sigma_j$ edges. Hence

$$|E(H_j)| \leq n^{1-t_{J-j+1}} n^{t_{J-j}-t_{J-j+1}}\delta_j \;=\; n^{1+t_{J-j}-2t_{J-j+1}}\delta_j \;. \tag{7}$$

Observe that the exponent is

$$1 + t_{J-j} - 2t_{J-j+1} \;=\; 1 + \frac{\kappa - (2^{J-j-1}-1)}{2^{J-j-1}\kappa} - 2\cdot\frac{\kappa-(2^{J-j}-1)}{2^{J-j}\kappa} \;=\; \frac{\kappa+1}{\kappa}\;.$$

Thus $|E(H_j)| \;=\; O(n^{1+1/\kappa}\Upsilon^{j/J})$. Hence the size of the subgraph H generated at the end of step 3 of Algorithm Sp_Cons can be bounded by

$$|E(H)| \;\leq\; \sum_{j=1}^{J}|E(H_j)| \;=\; O(n^{1+1/\kappa})\sum_{j=1}^{J}(\Upsilon^{1/J})^j \;=\; O(\Upsilon n^{1+1/\kappa})\;. \qquad \square$$

Lemma 3.9. *Step* 5 *of Algorithm* Sp_Cons *inserts at most* $O(\Upsilon n^{2/\kappa})$ *additional edges to the subgraph* H.

Proof. By Lemma 3.2,

$$\mathcal{R}_J \;\subseteq\; \mathcal{C}'_{J-1} \cup \bigcup_{i\in\tau_J}\mathcal{A}_i(\mathcal{G})\;,$$

and thus

$$\check{S}(\mathcal{R}_J) \;\geq\; \min\left\{\check{S}(\mathcal{C}'_{J-1}), \check{S}\left(\bigcup_{i\in\tau_J}\mathcal{A}_i(\mathcal{G})\right)\right\} \;=\; \min\left\{\check{S}(\mathcal{C}'_{J-1}), \min_{i\geq\kappa t_1}\check{S}(\mathcal{A}_i(\mathcal{G}))\right\}\;.$$

By Lemma 3.5, $\check{S}(\mathcal{C}'_{J-1}) \geq n^{t_2}n^{t_1-t_2} = n^{t_1}$. By Lemma 2.1, $\min_{i\geq\kappa t_1}\{\check{S}(\mathcal{A}_i(\mathcal{G}))\} \geq n^{t_1}$. Hence $\check{S}(\mathcal{R}_J) \geq n^{t_1}$. Thus the number of pairs of such clusters is no greater than $n^{2-2t_1} = n^{2/\kappa}$. Since the path inserted into the output subgraph between each pair is of length at most $2\cdot\Upsilon$, the total number of edges inserted is at most $O(\Upsilon n^{2/\kappa})$. $\square$

Combining Lemmas 3.8 and 3.9, we have the following.

Corollary 3.10. *The size of the subgraph output by Algorithm* Sp_Cons *is* $O(\Upsilon n^{1+1/\kappa})$.

3.3. Stretch analysis. In this section we bound the stretch of the spanner H output by Algorithm Sp_Cons. This is done by considering a pair of nodes $u, w \in V$, and one of the shortest paths P between u and w in G. This path is partitioned to segments of length no longer than Υ^J. It is convenient to visualize this path as going from left to right, from u to w (see Figure 5).

Consider some segment P' and let u' (resp., w') be its left (resp., right) endpoint. For a set X of clusters, a node u (resp., a path P) is said to be *X-clustered* if $u \in C$ (resp., $V(P) \subseteq C$) for some cluster C of X. Let u_J (resp., w_J) be the leftmost (resp., rightmost) $\mathcal{R}_J$-clustered node of P', and let C_{u_J} and C_{w_J} be the clusters such that $u_J \in C_{u_J}$ and $w_J \in C_{w_J}$. Since $dist_G(C_{u_J}, C_{w_J}) \leq dist_G(u_J, w_J) \leq \Upsilon$, there is a path of length $dist_G(C_{u_J}, C_{w_J})$ between some nodes $u'' \in C_{u_J}$ and $w'' \in C_{w_J}$ in the spanner H. It follows that there is a path of length no longer than $dist_G(u_J, w_J) + diam(C_{u_J}) + diam(C_{w_J})$ between u_J and w_J in H.

Next, we consider the subpaths from u to u_J, and from w_J from w for every segment of P, and observe that these subpaths are $(\mathcal{Z} \cup \bigcup_{i=1}^{J-1}\mathcal{R}_i)$-clustered. We partition these subpaths into subsegments of length $\Upsilon^{(J-1)/J}$, on each subsegment

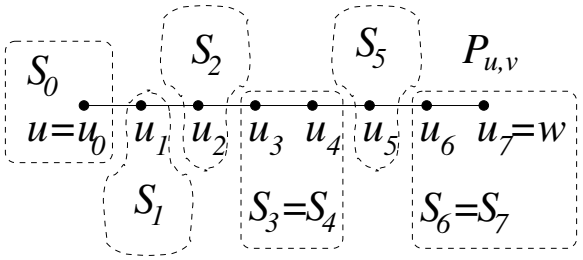


FIG. 5. *The path $P_{u,w}$ and the clusters along it.*

find the leftmost and the rightmost $\mathcal{R}_{J-1}$-clustered nodes and use the "short" paths between their clusters in the spanner. This argument is repeated recursively J times, until we are left with $\mathcal{Z}$-clustered subpaths, whose stretch can be easily bounded.

We next present a rigorous argument formalizing the above intuitive description. First, the following lemma can be proved using Lemmas 2.4 and 2.5 by a straightforward induction on the number of iterations of the algorithm.

LEMMA 3.11. *All the spanned partitions created throughout Algorithm* SP_CONS *are adjacency-preserving with respect to the resulting subgraph H returned by the algorithm.*

Next, we establish the following property of adjacency-preserving partitions that will be later used in the stretch analysis.

LEMMA 3.12. *Let $u, w \in V$ be a pair of nodes in the graph $G = (V, E)$, and let $P_{u,w} = (u = u_0, u_1, \ldots, u_x = w) \subseteq E$ be one of the shortest paths between them. Let H be a subgraph of G with $E(H) \subseteq E$, and let $\mathcal{S}$ be an adjacency-preserving spanned partition with respect to H, such that $V(P_{u,w}) \subseteq \bigcup_{(v,S,T)\in\mathcal{S}} S$. For $i = 0, 1, \ldots, x$, let $(v_i, S_i, T_i) \in \mathcal{S}$ be spanned clusters such that $u_i \in S_i$. (This is well-defined, since $V(P_{u,w}) \subseteq \bigcup_{(v,S,T)\in\mathcal{S}} S$, and the clusters of $\mathcal{S}$ are disjoint, by definition of spanned partition.) Finally, let $u'_0 \in S_0$, $u'_x \in S_x$ be some nodes. Then*

$$dist_H(u'_0, u'_x) \;\leq\; \sum_{i=0}^{x} (diam(v_i, S_i, T_i) + 1) - 1 \;.$$

Proof. The proof is by induction on x. The induction base is $x = 0$. Then $\{S_0, \ldots, S_x\} = \{S_0\}$, and $u = w \in S_0$. Let $u'_0, u'_x \in S_0$ be some arbitrary nodes in this cluster. Since $T_0 \subseteq H$, $dist_H(u'_0, u'_x) \leq diam(v_0, S_0, T_0)$, as required.

For the induction step, assume that the statement of the lemma is true for some $x \geq 0$. Consider the pair of neighboring clusters S_x, S_{x+1}. The edge (u_x, u_{x+1}) between them is spanned either through S_x or through S_{x+1}. In the former case, there exists a node $u'_x \in S_x$ such that the edge (u'_x, u_{x+1}) is in H. In the latter case, there exists a node u'_{x+1} such that the edge (u_x, u'_{x+1}) is in H.

Consider some pair of nodes $u''_0 \in S_0$, $u''_{x+1} \in S_{x+1}$. In the case when the edge (u_x, u_{x+1}) is spanned through S_x, using the induction hypothesis for the pair of nodes $u''_0 \in S_0$ and $u'_x \in S_x$, it follows that

$$\begin{aligned} dist_H(u''_0, u''_{x+1}) &\leq dist_H(u''_0, u'_x) + dist_H(u'_x, u_{x+1}) + dist_H(u_{x+1}, u''_{x+1}) \\ &\leq \left(\sum_{i=0}^{x} (diam(v_i, S_i, T_i) + 1) - 1 \right) + 1 + diam(v_{x+1}, S_{x+1}, T_{x+1}) \\ &= \sum_{i=0}^{x+1} (diam(v_i, S_i, T_i) + 1) - 1 \;. \end{aligned}$$

The case when the edge (u_x, u_{x+1}) is spanned through S_{x+1} follows symmetrically. □

We use the following notions. For $0 \leq j \leq J$, a path P in G is called a *class-j path* if it contains only $(\mathcal{Z} \cup \bigcup_{i=1}^{j} \mathcal{R}_i)$-clustered vertices (in particular, P is a class-0 path if all its vertices are $\mathcal{Z}$-clustered). A pair of vertices u, w is *j-reachable* if there is a shortest path between them that is a class-j path.

For every $1 \leq j \leq J$ denote

$$\gamma_j \;=\; \sum_{i=1}^{j} 2^i \cdot \hat{D}(\mathcal{R}_{j-i+1}).$$

For a real r, let $down(r) = \lceil r \rceil - 1$.

LEMMA 3.13. *For every integer $1 \leq j \leq J$, and for every j-reachable pair of vertices u'_j, u''_j,*

$$dist_H(u'_j, u''_j) \;\leq\; dist_G(u'_j, u''_j) \left(2 \cdot down(t_J \kappa) + 1 + \sum_{i=1}^{j} \frac{\gamma_i}{\Upsilon^{i/J}} \right) + \gamma_j \ .$$

Proof. Let P be a class-j path in G between u'_j and u''_j. It is convenient to visualize the path P as going from left to right, with the vertex u'_j at the leftmost end and the vertex u''_j the rightmost end. We prove by induction on j a claim that is slightly stronger than the statement of the lemma. Specifically, let S'_j, S''_j be the clusters such that $u'_j \in S'_j$, $u''_j \in S''_j$. Let $v'_j \in S'_j$, $v''_j \in S''_j$ be arbitrary nodes. Then

$$dist_H(v'_j, v''_j) \;\leq\; dist_G(u'_j, u''_j) \left(2 \cdot down(t_J \kappa) + 1 + \sum_{i=1}^{j} \frac{\gamma_i}{\Upsilon^{i/J}} \right) + \gamma_j \ .$$

Induction base ($j = 1$): We claim that

$$dist_H(v'_1, v''_1) \;\leq\; dist_G(u'_1, u''_1) \left(2 \cdot down(t_J \kappa) + 1 + \frac{\gamma_1}{\Upsilon^{1/J}} \right) + \gamma_1 \tag{8}$$

for a pair of 1-reachable vertices $u'_1 \in S'_1, u''_1 \in S''_1$ and any pair of nodes $v'_1 \in S'_1$, $v''_1 \in S''_1$.

We separate the discussion to two cases.

Case 1. $dist_G(u'_1, u''_1) \leq 2\Upsilon^{1/J}$.

Case 1.1. If the path P is a class-0 path, then it contains only $\mathcal{Z}$-clustered vertices. Note that $\mathcal{Z} = \bigcup_{i < t_J \kappa} \mathcal{A}_i(\mathcal{G}) = \bigcup_{i=0}^{down(t_J \kappa)} \mathcal{A}_i(\mathcal{G})$. Denote $z = down(t_J \kappa)$. By Lemmas 3.11 and 3.12,

$$\begin{aligned} dist_H(v'_1, v''_1) &\leq (dist_G(u'_1, u''_1) + 1) \left(\hat{D}(\mathcal{Z}) + 1 \right) - 1 \\ &\leq dist_G(u'_1, u''_1)(2z + 1) + 2z \ . \end{aligned}$$

Case 1.2. If the path P contains only one $\mathcal{R}_1$-clustered vertex, then by Lemmas 3.11 and 3.12,

$$dist_H(v'_1, v''_1) \;\leq\; dist_G(u'_1, u''_1)(2z + 1) + \hat{D}(\mathcal{R}_1) \ ,$$

and, again, we are done.

Case 1.3. It therefore remains to consider only the case where at least two vertices in the path P are $\mathcal{R}_1$-clustered. Let l_2 be the distance between the leftmost $\mathcal{R}_1$-clustered vertex, $w_1' \in S_1'$, and the rightmost $\mathcal{R}_1$-clustered vertex, $w_1'' \in S_1''$, of P. Hence the two subpaths, from v_1' to w_1' and from w_1'' to v_1'', are spanned with multiplicative stretch of at most $(\hat{D}(\mathcal{Z})+1)$. The distance between S_1' and S_1'' in G is at most $2\Upsilon^{1/J} = 2\delta_1(\Upsilon, J)$. Since at iteration 1 of Algorithm Sp_Cons, at step 4 of Procedure SC the shortest paths between all the pairs of clusters from $\mathcal{R}_1$ that are at distance at most $2\delta_1(\Upsilon, J)$ one from another were inserted into the subgraph H, it follows that there exist nodes $z_1' \in S_1'$, $z_1'' \in S_1''$ such that $dist_H(z_1', z_1'') = dist_G(w_1', w_1'') = l_2$. By Lemmas 3.11 and 3.12,

$$dist_H(v_1', z_1') \le dist_G(u_1', w_1')(2z+1) + \hat{D}(\mathcal{R}_1) \ ,$$
$$dist_H(z_1'', v_1'') \le dist_G(w_1'', u_1'')(2z+1) + \hat{D}(\mathcal{R}_1) \ .$$

Hence

$$\begin{aligned} dist_H(v_1', v_1'') &\le (dist_G(u_1', u_1'') - l_2)(2z+1) + 2\hat{D}(\mathcal{R}_1) + l_2 \\ &\le dist_G(u_1', u_1'')(2z+1) + 2\hat{D}(\mathcal{R}_1) \ . \end{aligned} \tag{9}$$

Case 2. $dist_G(u_1', u_1'') > 2\Upsilon^{1/J}$. By partitioning the path P into segments of length $\Upsilon^{1/J}$, we get $dist_G(u_1', u_1'') = (a-1)\Upsilon^{1/J} + \Upsilon'$, where $\Upsilon^{1/J} < \Upsilon' < 2\Upsilon^{1/J}$ and $a = \lfloor dist_G(u_1', u_1'')/\Upsilon^{1/J} \rfloor$. Applying the previous argument to each segment separately, it follows that

$$\begin{aligned} dist_H(v_1', v_1'') &\le (a-1)\left(\Upsilon^{1/J}(2z+1) + 2\hat{D}(\mathcal{R}_1)\right) + \Upsilon'(2z+1) + 2\hat{D}(\mathcal{R}_1) \\ &= dist_G(u_1', u_1'')(2z+1) + 2a \cdot \hat{D}(\mathcal{R}_1) \\ &\le dist_G(u_1', u_1'')(2z+1) + \frac{dist_G(u_1', u_1'') \cdot 2\hat{D}(\mathcal{R}_1)}{\Upsilon^{1/J}} \\ &= dist_G(u_1', u_1'')\left(2z+1+\frac{\gamma_1}{\Upsilon^{1/J}}\right) \ . \end{aligned} \tag{10}$$

As $\gamma_1 = 2\hat{D}(\mathcal{R}_1)$, the expression (8) dominates both (9) and (10), completing the proof of the induction base.

Induction step: Assume the induction hypothesis for some integer $1 < j \le J-1$. Consider a class-$(j+1)$ path P connecting u' and u'' in G. Let v' (resp., v'') be an arbitrary node in the same cluster as u' (resp., u'').

Case 1. P is of length no greater than $\Upsilon^{(j+1)/J}$.

We break the discussion into three subcases.

Case 1.1. If no $\mathcal{R}_{j+1}$-clustered vertex appears in the path P, we apply the induction hypothesis and we are done.

Case 1.2. Exactly one $\mathcal{R}_{j+1}$-clustered vertex w appears on P. Let w' be the left-hand neighbor of w and let w'' be the right-hand neighbor of w. Denote by S (resp., S'; S'') the cluster that contains w (resp., w'; w''). Let P' (resp., P'') be the path between u' and w' (resp., w'' and u''). Note that both subpaths P' and P'' are class-j paths (see Figure 6). Hence, the induction hypothesis is applicable to these subpaths.

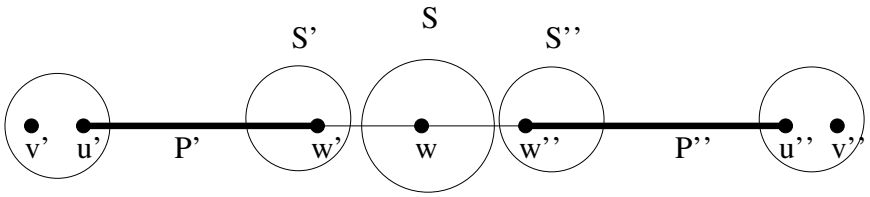


FIG. 6. *The solid lines represent class-j paths P' and P''. The cluster S is in $\mathcal{R}_{j+1}$.*

Therefore,

$$\begin{aligned} dist_H(v',v'') &\le dist_H(v',w') + dist_H(w',w'') + dist_H(w'',v'') \\ &\le (dist_G(u',u'')-2)\left(2z+1+\sum_{i=1}^{j}\frac{\gamma_i}{\Upsilon^{i/J}}\right) \\ &\quad +2\gamma_j + dist_H(w',w'') \ . \end{aligned}$$

The analysis decomposes again to three subsubcases.

Case 1.2.1. The first is that both edges (w',w) and (w,w'') are spanned through the cluster S, i.e., there exist nodes $z,y \in S$ such that the edges (w',z) and (y,w'') are in H. Then

$$\begin{aligned} dist_H(w',w'') &\le dist_H(w',z) + dist_H(z,y) + dist_H(y,w'') \\ &\le 2 + \hat{D}(\mathcal{R}_{j+1}) \ . \end{aligned}$$

Case 1.2.2. The edge (w',w) is spanned through the cluster S', and the edge (w,w'') is spanned through the cluster S''. In this situation there exist nodes $z' \in S'$, $z'' \in S''$ such that the edges (z',w) and (w,z'') are in H. Thus $dist_H(w',w'') \le dist_H(w',z') + dist_H(z',z'') + dist_H(z'',w'')$. However, we observe that the induction hypothesis is applicable to the pairs of nodes v',z' and z'',v'' as well. Note also that $dist_H(z',z'')=2$. Hence

$$\begin{aligned} dist_H(v',v'') &\le dist_H(v',z') + dist_H(z',z'') + dist_H(z'',v'') \\ &\le (dist_G(u',u'')-2)\left(2z+1+\sum_{i=1}^{j}\frac{\gamma_i}{\Upsilon^{i/J}}\right) + 2\gamma_j + 2 \ . \end{aligned}$$

Case 1.2.3. The edge (w',w) is spanned through the cluster S', and the edge (w,w'') is spanned through the cluster S (the situation when the edge (w',w) is spanned through the cluster S, and the edge (w,w'') is spanned through the cluster S'' is symmetrical to this one). In this subcase there exist nodes $z' \in S'$, $z \in S$ such that the edges (z',w) and (z,w'') are in H. Hence we may apply the induction hypothesis to the pairs v',z' and w'',v'', and get

$$\begin{aligned} dist_H(v',v'') &\le (dist_G(u',u'')-2)\left(2z+1+\sum_{i=1}^{j}\frac{\gamma_i}{\Upsilon^{i/J}}\right) \\ &\quad +2\gamma_j + 2 + \hat{D}(\mathcal{R}_{j+1}) \ . \end{aligned}$$

In all these three subsubcases,

$$dist_H(v',v'') \;\le\; dist_G(u',u'')\left(2z+1+\sum_{i=1}^{j}\frac{\gamma_i}{\Upsilon^{i/J}}\right)+2\gamma_j+\hat{D}(\mathcal{R}_{j+1})\ .$$

This expression is smaller than the bound in Lemma 3.13 with $j+1$ substituted for j, and so we are done in Case 1.2 too.

Case 1.3. There are at least two $\mathcal{R}_{j+1}$-clustered vertices in P. Let w (resp., z) be the leftmost (resp., rightmost) such vertex. Again, let w' (resp., z'') denote the left-hand (resp., right-hand) neighbor of w (resp., z). Denote by S_w, S_z, S', and S'' the clusters such that $w\in S_w$, $z\in S_z$, $w'\in S'$, and $z''\in S''$. Observe that H contains a path of length $dist_G(S_w,S_z)\le dist_G(w,z)$ between S_w and S_z. Similarly to the analysis of Case 1.2, where there was only one $\mathcal{R}_j$-clustered node in P, we decompose the analysis to subcases depending on whether the edge (w',w) is spanned through S_w or S', and on whether the edge (z,z'') is spanned through S_z or S''. Like in that situation, we apply the induction hypothesis on the subpaths between v' and w' (or some other appropriate node in S') and between z'' (or some other appropriate node in S'') and v''. Recall that $dist_G(S_w,S_z)\le 2\Upsilon^{(j+1)/J}=2\delta_{j+1}(\Upsilon,J)$. Also, at $(j+1)$st iteration of Algorithm Sp_Cons, at step 4 of Procedure SC (if $j<J-1$; if $j=J-1$ then at step 4 of Algorithm Sp_Cons) one of the shortest paths between S_w and S_z in G was inserted into H. Thus

$$\begin{aligned} dist_H(v',v'') &\le (dist_G(u',u'')-dist_G(w,z))\left(2z+1+\sum_{i=1}^{j}\frac{\gamma_i}{\Upsilon^{i/J}}\right)\\ &\quad+2\gamma_j+2\hat{D}(\mathcal{R}_{j+1})+dist_G(w,z)\\ &\le dist_G(u',u'')\left(2z+1+\sum_{i=1}^{j}\frac{\gamma_i}{\Upsilon^{i/J}}\right)+\gamma_{j+1}\ . \end{aligned}\tag{11}$$

Case 2. $dist_G(u',u'')>2\Upsilon^{(j+1)/J}$. In this situation, we partition the path into segments of length $\Upsilon^{(j+1)/J}$ each, except the last segment which may be of length between $\Upsilon^{(j+1)/J}$ and $2\Upsilon^{(j+1)/J}$. We next show that in this situation γ_{j+1} divided by $\Upsilon^{(j+1)/J}$ is introduced into the multiplicative term. Formally, $dist_G(u',u'')=(a-1)\Upsilon^{(j+1)/J}+\Upsilon'$, where $\Upsilon^{(j+1)/J}<\Upsilon'<2\Upsilon^{(j+1)/J}$ and $a=\lfloor dist_G(u_1',u_1'')/\Upsilon^{1/J}\rfloor$. Then

$$\begin{aligned} dist_H(v',v'') &\le (a-1)\left(\left(2z+1+\sum_{i=1}^{j}\frac{\gamma_i}{\Upsilon^{i/J}}\right)\Upsilon^{(j+1)/J}+\gamma_{j+1}\right)\\ &\quad+\left(2z+1+\sum_{i=1}^{j}\frac{\gamma_i}{\Upsilon^{i/J}}\right)\Upsilon'+\gamma_{j+1}\\ &=\left(2z+1+\sum_{i=1}^{j}\frac{\gamma_i}{\Upsilon^{i/J}}\right)\left((a-1)\Upsilon^{(j+1)/J}+\Upsilon'\right)+\gamma_{j+1}a\\ &\le dist_G(u',u'')\left(2z+1+\sum_{i=1}^{j}\frac{\gamma_i}{\Upsilon^{i/J}}\right)+\gamma_{j+1}\ \frac{dist_G(u',u'')}{\Upsilon^{(j+1)/J}}\\ &= dist_G(u',u'')\left(2z+1+\sum_{i=1}^{j+1}\frac{\gamma_i}{\Upsilon^{i/J}}\right)\ . \end{aligned}\tag{12}$$

In summary, for any $(j+1)$-reachable pair of vertices $u' \in S'$, $u'' \in S''$, and for any pair of nodes $v' \in S'$, $v'' \in S''$ the bound $dist_H(v', v'') \leq dist_G(u', u'')(2z+1+\sum_{i=1}^{j+1} \frac{\gamma_i}{\Upsilon^{i/J}}) + \gamma_{j+1}$ dominates both former bounds (11) and (12) on $dist_H(v', v'')$. □

Observe that any path in G is a class-J path. Hence by Lemma 3.13 we have the following corollary.

COROLLARY 3.14. *For any pair of vertices u', u'', $dist_H(u', u'') \leq \alpha \cdot dist_G(u', u'') + \beta$ for $\alpha = 2 \cdot down(t_J\kappa) + 1 + \sum_{i=1}^{J} \frac{\gamma_i}{\Upsilon^{i/J}}$ and $\beta = \gamma_J$.*

LEMMA 3.15. *For Υ and J such that $\Upsilon^{1/J} \geq 6$ and for any $1 \leq j \leq J$, $\gamma_j = O(t_{J-1}\kappa \cdot 3^j + \Upsilon^{(j-1)/J})$.*

Proof. By the definition of γ_j, $\gamma_j = \sum_{i=0}^{j-1} 2^{j-i} \hat{D}(\mathcal{R}_{i+1})$ for any $1 \leq j \leq J$. Next substitute the explicit formula for $\hat{D}(\mathcal{R}_j)$ given in Corollary 3.4 and get (dividing the expression by 4 for convenience)

$$\frac{\gamma_j}{4} \leq 2^{j-1} t_{J-1}\kappa + 2^{j-2}\left(3t_{J-1}\kappa + \Upsilon^{1/J}\right) + \cdots + 2^0\left(3^{j-1}\kappa t_{J-1} + \sum_{i=1}^{j-1} 3^{j-1-i}\Upsilon^{i/J}\right)$$

$$= t_{J-1}\kappa \sum_{i=0}^{j-1} 3^i 2^{j-1-i} + \sum_{l=1}^{j-1}\left(\Upsilon^{l/J} \sum_{i=0}^{j-l-1} 3^i 2^{j-l-1-i}\right). \tag{13}$$

Note that for any integer $p \geq 1$, $\sum_{i=0}^{p} 3^i 2^{p-i} < 3^p \frac{1}{1-2/3} = 3^{p+1}$. Since $\Upsilon^{1/J} \geq 6$,

$$\begin{aligned} \gamma_j/4 &\leq t_{J-1}\kappa 3^j + \sum_{i=1}^{j-1} \Upsilon^{i/J} 3^{j-i} \leq t_{J-1}\kappa 3^j + \Upsilon^{(j-1)/J} 3 \sum_{i=0}^{j-2}\left(\frac{3}{\Upsilon^{1/J}}\right)^i \\ &< t_{J-1}\kappa 3^j + 6\Upsilon^{(j-1)/J}. \end{aligned}$$ □

LEMMA 3.16. *For Υ and J such that $\Upsilon^{1/J} \geq 6$,*

1. *the additive term is $\beta = O(t_{J-1}\kappa \cdot 3^J + \Upsilon^{(J-1)/J})$,*
2. *the multiplicative factor is bounded by $\alpha = 1 + 2 \cdot down(t_J\kappa) + O(\frac{J + t_{J-1}\kappa}{\Upsilon^{1/J}})$.*

Proof. By Corollary 3.14 and substituting $j = J$ in Lemma 3.15, we get

$$\begin{aligned} \alpha &\leq 1 + 2 \cdot down(t_J\kappa) + \sum_{j=1}^{J} \frac{\gamma_j}{\Upsilon^{j/J}} \\ &\leq 1 + 2 \cdot down(t_J\kappa) + O\left(\frac{J}{\Upsilon^{1/J}}\right) + O\left(t_{J-1}\kappa \sum_{j=1}^{J} \frac{3^j}{\Upsilon^{j/J}}\right) \\ &\leq 1 + 2 \cdot down(t_J\kappa) + O\left(\frac{J}{\Upsilon^{1/J}}\right) + O\left(t_{J-1}\kappa \cdot \frac{1}{\Upsilon^{1/J}} \cdot \frac{1}{1 - 3/\Upsilon^{1/J}}\right) \\ &= 1 + 2 \cdot down(t_J\kappa) + O\left(\frac{J + t_{J-1}\kappa}{\Upsilon^{1/J}}\right) \end{aligned}$$

and $\beta = O(t_{J-1}\kappa 3^J + \Upsilon^{(J-1)/J})$. This completes the proof of Lemma 3.16. □

THEOREM 3.17. *For any fixed $0 < \epsilon < 1$ and fixed integer $2 \leq \kappa = O(\log n)$ there exists a fixed $\beta = \beta(\kappa, \epsilon) = \kappa^{\max\{\log\log\kappa - \log\epsilon, 3\}}$ such that for any graph G, running Algorithm* SP_CONS *on G, κ, $J = \log\kappa$ and β yields a $(1+\epsilon, \beta)$-spanner of G with $O(\beta n^{1+1/\kappa})$ edges.*

Proof. Since $t_i = (\kappa - 2^{i-1}) \,/\, (2^{i-1}\kappa)$, it is sufficient to set $J = \lceil \log \kappa \rceil$ in order to ensure $t_J \kappa \leq 1$, i.e., $down(t_J \kappa) = 0$. Therefore, substituting $J = \lceil \log \kappa \rceil$ into Lemma 3.16 implies that the multiplicative factor of the stretch is $\alpha = 1 + O(\frac{\log \kappa}{\Upsilon^{1/\log \kappa}})$ and the additive term is $\beta = O(\Upsilon^{(\log \kappa - 1)/\log \kappa} + \kappa^{\log 3})$. To get a multiplicative stretch of $1+\epsilon$ for any fixed $0 < \epsilon < 1$ and to ensure that $\Upsilon^{1/J} \geq \kappa/2^{J-3} = 8$, we set $\Upsilon^{1/\log \kappa} = \max\{\epsilon^{-1} \log \kappa, 8\}$, or $\Upsilon = \kappa^{\max\{\log\log\kappa - \log\epsilon, 3\}}$. The theorem now follows by Corollary 3.10. □

Note that for $\kappa = \Theta(\log n)$ the size of the spanner becomes $O(\beta(\kappa,\epsilon)n) = o(n^{1+\nu})$ for any $\nu > 0$, and so there is no reason to consider values of κ greater than $O(\log n)$.

COROLLARY 3.18. *For any constant $\epsilon > 0$, $\lambda > 0$ there exists a constant $\beta'(\epsilon,\lambda)$ such that for any n-vertex graph $G = (V,E)$ there exists a polynomial time constructible subgraph H, $E(H) \subseteq E$, with $O(n^{1+\lambda})$ edges, such that for every pair of vertices $u, w \in V$ with $dist_G(u,w) \geq \beta'(\epsilon,\lambda)$, $dist_H(u,w) \leq (1+\epsilon) dist_G(u,w)$.*

Proof. By Theorem 3.17 there exists a constant $\beta(\epsilon/2, \lceil 1/\lambda \rceil)$ such that there exists a polynomial time constructible $(1+\epsilon/2, \beta(\epsilon/2, \lceil 1/\lambda \rceil))$-spanner H, $E(H) \subseteq E$, with $O(n^{1+\lambda})$ edges. Set $\beta'(\epsilon,\lambda) = 2\beta(\epsilon/2, \lceil 1/\lambda \rceil) \,/\, \epsilon$ and observe that for any pair of vertices $u, w \in V$ with $dist_G(u,w) \geq \beta'(\epsilon,\lambda)$,

$$dist_H(u,w) \;\leq\; (1+\epsilon/2) dist_G(u,w) + \beta(\epsilon/2, \lceil 1/\lambda \rceil) \;\leq\; (1+\epsilon) dist_G(u,w)\,. \qquad \square$$

In order to get a multiplicative stretch factor asymptotically close to 1 we just substitute $\epsilon = 1/\log^b n$ for any constant b in Theorem 3.17.

COROLLARY 3.19. *For any fixed integer $2 \leq \kappa = O(\log n)$ and constant $b > 0$ and for any graph G, running Algorithm* SP_CONS *with G, κ, $\beta(\kappa,n) = \kappa^{\log\log\kappa} \log^{b \log \kappa} n$ and $J = \log \kappa$ yields a $(1 + 1/\log^b n, \beta(\kappa,n))$-spanner of G with $O(\beta(\kappa,n) \cdot n^{1+1/\kappa})$ edges.*

Again, for any constant $\kappa \geq 2$ this yields a $(1 + 1/\operatorname{polylog} n, \operatorname{polylog} n)$-spanner with $\tilde{O}(n^{1+1/\kappa})$ edges.

In order to get an almost linear number of edges (or formally, $o(n^{1+\nu})$ for any $\nu > 0$) we substitute $\epsilon = (\log n)^{\log^{(3)} n}$ and $\kappa = \log n$ in Theorem 3.17, and get the following.

COROLLARY 3.20. *For any graph G, running Algorithm* SP_CONS *with G, $\kappa = \log n$, $\beta(n) = (\log n)^{O(\log^{(2)} n \log^{(3)} n)}$ and $J = \log \kappa$ yields a $(1+1/(\log n)^{\log^{(3)} n}, \beta(n))$-spanner of G with $O(\beta(n)n)$ edges.*

Theorem 3.17 also enables us to decrease the multiplicative stretch to $1 + 2^{-\log n/\log^{(b)} n}$ while not significantly increasing the other parameters. Specifically, we obtain the following.

COROLLARY 3.21. *For any fixed $2 \leq \kappa = O(\log n)$, constant $b \geq 2$ and graph G, running Algorithm* SP_CONS *with G, κ, $J = \log \kappa$ and $\beta(\kappa,n) = \kappa^{\log\log\kappa} \cdot 2^{\log\kappa \log n/\log^{(b)} n}$ yields a $(1 + 2^{-\log n/\log^{(b)} n}, \beta(\kappa,n))$-spanner of G with $O(\beta(\kappa, n)n^{1+\frac{1}{\kappa}})$ edges.*

In particular, for $\kappa = \log n$ this yields a $(1 + 2^{-\log n/\log^{(b)} n}, 2^{O(\log n/\log^{(b)} n)})$-spanner with $2^{O(\log n/\log^{(b)} n)} n$ edges.

Finally, we remark that for some specific small values of κ, tighter bounds on ϵ and β parameters of the spanner that is constructed by Algorithm SP_CONS, can be obtained (see the preliminary versions of this paper [9, 10] for the details). In particular, the following statements hold.

THEOREM 3.22.

1. *Algorithm* SP_CONS*, when run with parameter* $\kappa = 2$*, yields a construction of an additive* 2*-spanner with* $O(n^{3/2})$ *edges.*
2. *Algorithm* SP_CONS*, when run with parameter* $\kappa = 3$*, yields a construction of a* $(1+\epsilon, 4)$*-spanner with* $O(\epsilon^{-1}n^{4/3})$ *edges for any* $\epsilon > 0$*.*

The first result improves by a logarithmic factor the result of [7] and it is tight up to constant factors due to an $\Omega(n^{3/2})$ lower bound of [16]. However, the running time of Algorithm SP_CONS with parameter $\kappa = 2$ is $O(n^{5/2})$, which is significantly larger than the running time of the algorithm of [7], which is $\tilde{O}(n^2)$. The second result significantly improves the previously known construction of 5-spanner with $O(n^{4/3})$ edges, due to [1].

4. Running time. In this section we analyze the running time of Algorithm SP_CONS. We then show that it can be modified to run in time $\tilde{O}(n^{2+\mu})$, for arbitrarily small $\mu > 0$, while maintaining α, β and the size of the generated spanner as before.

4.1. Time complexity of Algorithm SP_CONS. We start by analyzing the running time of Algorithm SP_CONS as described above, without modifications.

LEMMA 4.1. *The running time of Procedure* DOWN_PART *is* $O(|E|+n^{1+1/\kappa}\log n)$.

Proof. Procedure DOWN_PART starts by picking a vertex and running a depth-limited version of the unweighted BFS algorithm from this vertex. The BFS algorithm continues adding new layers until the next iteration increases the size of the cluster by a factor smaller than $n^{1/\kappa}$. Let S be some cluster of the spanned partition $\mathcal{G}$ built by the procedure. By using appropriate data structures, the operation of counting the number of vertices at the next layer that joins the cluster takes $O(|S|n^{1/\kappa}\log(|S|n^{1/\kappa}))$ time, since the total number of vertices involved in this process is $O(|S|n^{1/\kappa})$. Hence summing over all the cluster constructions, the running time invested in deciding whether to terminate the construction of the current cluster and to start the construction of the next cluster is

$$\sum_{S\in\mathcal{G}} O(|S|n^{1/\kappa}\log(|S|n^{1/\kappa})) \;=\; O\left(n^{1/\kappa}\log n\sum_{S\in\mathcal{G}}|S|\right) \;=\; O(\log n\cdot n^{1+1/\kappa})\ .$$

Note that when constructing a new cluster, Procedure DOWN_PART might touch the vertices of the shell of an already built cluster S, but it will never explore again the edges that have at least one endpoint in S. Note also that the edges connecting two vertices of the shell of S were not explored during the construction of the cluster S. Thus Procedure DOWN_PART never re-explores edges that were previously explored.

For a cluster $S_i \in \mathcal{G}$, let $E'(S_i)$ be the set of edges explored during the construction of the cluster S_i. By the above considerations, $E'(S_i)\cap E'(S_j) = \emptyset$ for any two clusters $S_i, S_j \in \mathcal{G}$. Also $\cup_{S\in\mathcal{G}}E'(S) \subseteq E$. The depth-limited unweighted BFS algorithm that explores m' edges and n' vertices requires $O(m' + n'\log n')$ time. Hence the total running time of all invocations of the depth-limited unweighted BFS algorithm by Procedure DOWN_PART is $\sum_{S\in\mathcal{G}}|E'(S)| + O(n^{1+1/\kappa}\log n) = O(|E| + n^{1+1/\kappa}\log n)$. Hence the overall running time of the procedure is $O(|E| + n^{1+1/\kappa}\log n)$. □

Note that the problem is interesting mainly when $|E|$ is greater than $O(n^{1+1/\kappa})$, since otherwise the graph itself may serve as its own sparse 1-spanner, with $O(n^{1+1/\kappa})$ edges.

LEMMA 4.2. *Procedure* SC *can be executed on an input spanned partition* $\mathcal{C}$ *in* $O(|E|n/\check{S}(\mathcal{C}))$ *time.*

Proof. It is easy to see that the most expensive parts of Procedure SC are steps 2(b)iiA and 4, which are concerned with computing the shortest paths between clusters. This task can be performed by running BFS algorithm separately from every cluster. Note that BFS algorithm enables computing all the distances from a single source, where this source need not be a single vertex but may be a subset of vertices as well. Recall that for a subset of vertices U and a vertex v, the distance between U and v is $\min\{dist_G(u,v) \mid u \in U\}$. Each invocation of BFS algorithm requires $O(|E| + n \cdot \log n) = O(|E|)$ time. The number of clusters is $O(n/\check{S}(\mathcal{C}))$. Hence the running time of the procedure is $O(|E|n/\check{S}(\mathcal{C}))$. ∎

LEMMA 4.3. *The running time of Algorithm* SP_CONS *is* $O(|E|n^{(\kappa-1)/\kappa})$.

Proof. The dominant term in the time complexity of the algorithm is the running time of the very first invocation of Procedure SC. In this invocation, $\check{S}(\mathcal{G}) \geq n^{1/\kappa}$, and so by Lemma 4.2 it requires $O(|E|n^{(\kappa-1)/\kappa})$ time. The running time of all later invocations of SC is significantly smaller, as those are applied to spanned partitions $\mathcal{C}$ with $\check{S}(\mathcal{C}) \geq n^{2/\kappa}$. The complexity of step 5 can be analyzed along the same lines as in the proof of Lemma 4.2 and shown to be at most $O(|E|n^{1/\kappa})$. The running time of Procedure DOWN_PART is at most $O(|E| + n^{1+1/\kappa}\log n)$. Thus, the overall complexity of Algorithm SP_CONS is $O(|E|n^{(\kappa-1)/\kappa})$. ∎

4.2. Speeding up algorithm SP_CONS. In this section we present a modification of Algorithm SP_CONS that has smaller time complexity, but α, β and size parameters similar to those of section 3.3.

The main idea is to save time by finding *almost* shortest paths instead of shortest ones. Specifically, the modified algorithm receives an additional parameter $t \geq 1$. It starts with invoking an *all pairs almost shortest path* algorithm $APASP_t$ due to [7]. This algorithm runs for $\tilde{O}(n^{2+1/t})$ time and for every pair of vertices $u, w \in V$ computes a path of length $dist'(u,w) \leq dist_G(u,w) + t$. Next, the algorithm sorts the obtained n^2 distances in $O(n^2 \log n)$ time and for each pair maintains the index in the sorted array (i.e., a pair of closer vertices will have a smaller index). Next, it runs Algorithm SP_CONS as is, except that whenever it needs to compute a shortest path between two clusters it uses the precomputed array. Specifically, for two clusters S_i and S_j it takes $|S_i| \cdot |S_j|$ time to find the pair of vertices $u_i \in S_i$, $u_j \in S_j$ such that $dist'(u_i,u_j) = \min\{dist'(u,w) \mid u \in S_i, w \in S_j\}$. Hence, overall, the approximate computation of all the distances between clusters of some partition $\mathcal{C}$ and all the paths between close cluster pairs (i.e., at distance bounded by $\Upsilon^{i/J}$ for some $1 \leq i \leq J$) takes time bounded by

$$O\left(\sum_{S_i,S_j\in\mathcal{C}} \Upsilon|S_i|\cdot|S_j|\right) \;\leq\; O\left(\Upsilon\left(\sum_{S_i\in\mathcal{C}}|S_i|\right)^2\right) \;=\; O(\Upsilon n^2)\ .$$

Since there are $O(J)$ iterations, the distances and paths for different partitions are computed $O(J)$ times, i.e., the total running time of computing the distances and paths is $O(J\Upsilon n^2)$. Hence, the overall running time of the modified algorithm is $\tilde{O}(n^{2+1/t} + J\Upsilon n^2)$, and setting $J = \log\kappa = O(\log\log n)$ and $\Upsilon = O(\kappa^{\log\kappa}) = O((\log n)^{\log\log n})$, the resulting time bound is $\tilde{O}(n^{2+1/t})$.

It remains to analyze the α, β and size parameters of the spanner obtained by the above modification of Algorithm SP_CONS. First, instead of Lemma 3.1 we now get

$$\hat{D}(\mathcal{C}') \;\leq\; 3\hat{D}(\mathcal{C}) + 2\delta + 2t. \tag{14}$$

Next, Lemma 3.3 is replaced by the following.

LEMMA 4.4. *For every integer* $1 \leq j \leq J-1$,

$$\text{(a)} \quad \hat{D}(\mathcal{C}'_j) \leq 2\left(3^j \kappa t_{J-1} + \sum_{l=1}^{j} 3^{j-l}\Upsilon^{l/J}\right) + t(3^j - 1) \ ,$$

$$\text{(b)} \quad \hat{D}(\mathcal{R}_j) \leq 2\left(3^{j-1} \kappa t_{J-1} + \sum_{l=1}^{j-1} 3^{j-1-l}\Upsilon^{l/J}\right) + t(3^{j-1} - 1) \ .$$

Proof. The induction base holds since $\hat{D}(\mathcal{R}_1) \leq \hat{D}(\mathcal{C}_1) \leq 2t_{J-1}\kappa + t(3^0 - 1) = 2t_{J-1}\kappa$ and $\hat{D}(\mathcal{C}_1) \leq 2(3^1 t_{J-1}\kappa + \Upsilon^{1/J}) + t(3^1 - 1)$, by Lemma 3.3 and inequality (14).

The induction hypothesis changes to

$$\hat{D}(\mathcal{C}'_j) \ \leq \ 2\left(3^j \kappa t_{J-1} + \sum_{l=1}^{j} 3^{j-l}\Upsilon^{l/J}\right) + t(3^j - 1) \ .$$

The inequalities (4) and (5) are unchanged, and hence

$$\hat{D}(\mathcal{R}_{j+1}) \ \leq \ \hat{D}(\mathcal{C}_{j+1}) \ \leq \ 2\left(3^j \kappa t_{J-1} + \sum_{l=1}^{j} 3^{j-l}\Upsilon^{l/J}\right) + t(3^j - 1) \ .$$

So by inequality (14), it follows that

$$\begin{aligned}\hat{D}(\mathcal{C}_{j+1}) &\leq 3 \cdot 2\left(3^j \kappa t_{J-1} + \sum_{l=1}^{j} 3^{j-l}\Upsilon^{l/J}\right) + 3t(3^j - 1) + 2\Upsilon^{(j+1)/J} + 2t \\ &= 2\left(3^{j+1} \kappa t_{J-1} + \sum_{l=1}^{j+1} 3^{j+1-l}\Upsilon^{l/J}\right) + t(3^{j+1} - 1) \ . \qquad \square\end{aligned}$$

Analogously to inequality (6), it follows that

$$\hat{D}(R_J) \ \leq \ 2\left(3^{J-1} \kappa t_{J-1} + \sum_{l=1}^{J-1} 3^{J-l-1}\Upsilon^{l/J}\right) + t(3^{J-1} - 1) \ .$$

Next, it is easy to see that inequality (7) becomes

$$|E(H_j)| \ = \ O(n^{1+t_{J-j}-2t_{J-j+1}}(\delta_{J+1-j} + t)) \ .$$

Hence

$$|E(H)| \ = \ \sum_{j=1}^{J-1} |E(H_j)| \ = \ O(n^{1+1/\kappa}(\Upsilon + tJ)) \ .$$

Lemmas 3.13 and 3.14 are unchanged. However, since the expression for $\hat{D}(\mathcal{R}_j)$ is modified, inequality (13) becomes

$$\begin{aligned}\gamma_j/4 &\leq 2^{j-1} t_{J-1}\kappa + 2^{j-2}\left(3t_{J-1}\kappa + \Upsilon^{1/J} + 2t\right) + \cdots \\ &\quad + 2^0\left(3^{j-1} t_{J-1}\kappa + \sum_{i=1}^{j-1} 3^{j-1-i}\Upsilon^{i/J} + t(3^{j-1} - 1)\right) \\ &= t_{J-1}\kappa \sum_{i=0}^{j-1} 3^i 2^{j-1-i} + \sum_{l=1}^{j-1}\left(\Upsilon^{l/J} \sum_{i=0}^{j-l-1} 3^i 2^{j-l-i-1}\right) + t\sum_{i=1}^{j-1} 2^{j-1-i}(3^i - 1) \\ &\leq t_{J-1}\kappa 3^j + 6\Upsilon^{(j-1)/J} + t3^j \ = \ 3^j(t_{J-1}\kappa + t) + 6\Upsilon^{(j-1)/J} \ .\end{aligned}$$

Hence the multiplicative factor is at most $1 + 2t_J\kappa + O(\frac{J + t_{J-1}\kappa + t}{\Upsilon^{1/J}})$, the additive term is

$O((t_{J-1}\kappa+t)3^J+\Upsilon^{(J-1)/J})$ and the size of the constructed spanner is $O(n^{1+1/\kappa}(\Upsilon+tJ))$, and we have the following.

THEOREM 4.5. *For any fixed $0<\epsilon<1$ and fixed integers $t\geq 1$ and $\kappa\geq 2$ there exist fixed $\bar{\beta}=\bar{\beta}(\kappa,\epsilon,t)=\kappa^{\max\{\log\log\kappa-\log\epsilon,3\}}+t\log\kappa$ and $\tilde{\beta}=\tilde{\beta}(\kappa,\epsilon,t)=\kappa^{\max\{\log\log\kappa-\log\epsilon,\log t,3\}}$ such that every n-vertex graph has a $(1+\epsilon,\tilde{\beta})$-spanner of size bounded by $O(n^{1+1/\kappa}\bar{\beta})$ that can be built in time $\tilde{O}(n^{2+1/t})$.*

In particular, setting $t=O(\log\kappa)$ we obtain results analogous to Theorem 3.17 and Corollaries 3.18–3.20 by an algorithm of time complexity $\tilde{O}(n^{2+1/\log\kappa})$. Getting a running time of $\tilde{O}(n^{2+1/\kappa})$ requires raising the additive term to $O(\kappa^{\max\{\log\kappa,-\log\epsilon\}})$, which is, nonetheless, still constant for constant κ and ϵ. This enables us to generalize Corollary 3.18 and get the following.

COROLLARY 4.6. *For any constant $\epsilon>0,\lambda>0,\mu>0$ there exists a constant $\beta''(\epsilon,\lambda,\mu)$ such that for any n-vertex graph $G=(V,E)$ there exists a subgraph H, $E(H)\subseteq E$, that satisfies the following properties*

1. *For every pair of vertices $u,w\in V$ with $dist_G(u,w)\geq\beta''(\epsilon,\lambda,\mu)$, $dist_H(u,w)\leq(1+\epsilon)dist_G(u,w)$.*
2. *$|E(H)|=O(n^{1+\lambda})$.*
3. *H can be constructed in $\tilde{O}(n^{2+\mu})$ time.*

Acknowledgments. We thank Uri Zwick for his helpful comments, corrections and suggestions of improvements, and an anonymous referee for helpful remarks.

REFERENCES

[1] I. ALTHÖFER, G. DAS, D. DOBKIN, D. JOSEPH, AND J. SOARES, *On sparse spanners of weighted graphs*, Discrete Comput. Geom., 9 (1993), pp. 81–100.
[2] B. AWERBUCH AND D. PELEG, *Sparse partitions*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, 1990, pp. 503–513.
[3] B. BOLLOBAS, D. COPPERSMITH, AND M. L. ELKIN, *Sparse subgraphs that preserve long distances and additive spanners*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003.
[4] B. CHANDRA, G. DAS, G. NARASIMHAN, AND J. SOARES, *New sparseness results on graph spanners*, in Proceedings of the 8th Annual ACM Symposium on Computational Geometry, Berlin, 1992, pp. 192–201.
[5] L. P. CHEW, *There is a planar graph almost as good as the complete graph*, in Proceedings of the 2nd Annual Symposium on Computational Geometry, 1986, pp. 169–177.
[6] D. P. DOBKIN, S. J. FRIEDMAN, AND K. J. SUPOWIT, *Delaunay graphs are almost as good as complete graphs*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 20–26.
[7] D. DOR, S. HALPERIN, AND U. ZWICK, *All pairs almost shortest paths*, SIAM J. Comput., 29 (2000), pp. 1740–1759.
[8] M. L. ELKIN, *Computing Almost Shortest Paths*, in Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing, Newport, RI, 2001, pp. 53–63.
[9] M. L. ELKIN AND D. PELEG, *$(1+\epsilon,\beta)$-Spanner Constructions for General Graphs*, Technical Report MCS00-17, Weizmann Institute of Science, Rehovot, Israel, 2000.
[10] M. L. ELKIN AND D. PELEG, *$(1+\epsilon,\beta)$-Spanner constructions for general graphs*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Crete, Greece, 2001.
[11] A. L. LIESTMAN AND T. C. SHERMER, *Additive Graph Spanners*, Tech. Report 91-5, Simon Fraser University, Burnaby, BC, Canada, 1991.
[12] A. L. LIESTMAN AND T. C. SHERMER, *Grid spanners*, Networks, 23 (1993), pp. 123–133.
[13] D. PELEG, *Distributed Computing: A Locality-Sensitive Approach*, SIAM, Philadelphia, PA, 2000.
[14] D. PELEG AND A. SCHÄFFER, *Graph spanners*, J. Graph Theory, 13 (1989), pp. 99–116.
[15] D. PELEG AND J. D. ULLMAN, *An optimal synchronizer for the hypercube*, SIAM J. Comput., 18 (1989), pp. 740–747.
[16] R. WENGER, *Extremal graphs with no C^4's, C^6's and C^{10}'s*, J. Combin. Theory Ser. B, 52 (1991), pp. 113–116.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 632–646

OPT VERSUS *LOAD* IN DYNAMIC STORAGE ALLOCATION*

ADAM L. BUCHSBAUM†, HOWARD KARLOFF†, CLAIRE KENYON‡, NICK REINGOLD†, AND MIKKEL THORUP†

Abstract. Dynamic storage allocation is the problem of packing given axis-aligned rectangles into a horizontal strip of minimum height by sliding the rectangles vertically but not horizontally. Where $L = LOAD$ is the maximum sum of heights of rectangles that intersect any vertical line and OPT is the minimum height of the enclosing strip, it is obvious that $OPT \geq LOAD$; previous work showed that $OPT \leq 3 \cdot LOAD$. We continue the study of the relationship between OPT and $LOAD$, proving that $OPT = L + O((h_{\max}/L)^{1/7})L$, where $h_{\max}$ is the maximum job height. Conversely, we prove that for any $\epsilon > 0$, there exists a $c > 0$ such that for all sufficiently large integers $h_{\max}$, there is a dynamic storage allocation instance with maximum job height $h_{\max}$, maximum load at most L, and $OPT \geq L + c(h_{\max}/L)^{1/2+\epsilon}L$, for infinitely many integers L. En route, we construct several new polynomial-time approximation algorithms for dynamic storage allocation, including a $(2+\epsilon)$-approximation algorithm for the general case and polynomial-time approximation schemes for several natural special cases.

Key words. approximation algorithms, dynamic storage allocation, polynomial-time approximation schemes

AMS subject classifications. 68Q17, 68Q25, 68W25, 68W40

DOI. 10.1137/S0097539703423941

1. Introduction.

We study a simple rectangle-packing problem: Given a set of n isothetic (i.e., axis-parallel) open rectangles in the x-y plane, the ith extending from x-coordinate x_i to x-coordinate y_i on the real line and having height h_i, slide each rectangle up or down *but not sideways* so that (1) each rectangle is a subset of the positive quadrant, (2) the regions of the plane they occupy are pairwise disjoint, and (3) the supremum of the y-coordinates used is minimized. Resembling off-line Tetris in which the rectangles slide vertically but not horizontally, this problem is formally known as dynamic storage allocation for the following reason. View a rectangle starting at x-coordinate x_i, ending at x-coordinate y_i, and of height h_i as a request for h_i contiguous bytes of storage starting at time x_i and ending at time y_i. Then the problem of finding a rectangle placement that minimizes the supremum of the y-coordinates used is exactly that of minimizing the number of contiguous bytes needed to satisfy all memory requests.

Formally, an instance of dynamic storage allocation is a set of some number n of *jobs*, each job i being an open interval $I_i = (x_i, y_i)$ (for rationals $x_i < y_i$), which we assume without loss of generality is a subset of $(0,1)$, and a positive rational *height* h_i. We say job i is *live at x-coordinate t* (or simply *live at t*) if $t \in I_i$. A feasible solution is an assignment of a nonnegative real $s(i)$ to each job i such that if we define $S(i)$ to be the open real interval $(s(i), s(i) + h_i)$ (the region of memory assigned to job i during time period (x_i, y_i)), then for all $t \in (0,1)$, the jobs i that are live at t have pairwise disjoint $S(i)$'s. The goal is to minimize the *makespan*: $\max_i\{s(i) + h_i\}$.

*Received by the editors March 11, 2003; accepted for publication (in revised form) September 10, 2003; published electronically March 30, 2004. An extended abstract appears in *Proceedings of the 35th ACM Symposium on Theory of Computing*, 2003.
http://www.siam.org/journals/sicomp/33-3/42394.html

†AT&T Labs, Shannon Laboratory, 180 Park Ave., Florham Park, NJ 07932 (alb@research.att.com, howard@research.att.com, reingold@research.att.com, mthorup@research.att.com).

‡Laboratoire d'Informatique, Ecole Polytechnique, 91128 Palaiseau Cedex, France (kenyon@lix.polytechnique.fr).

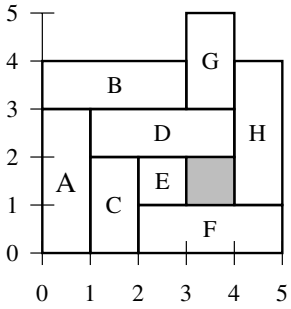


FIG. 1.1. *An optimal packing for the jobs (in the form* (x_i, y_i, h_i)*)* $A = (0, 1, 3)$, $B = (0, 3, 1)$, $C = (1, 2, 2)$, $D = (1, 4, 1)$, $E = (2, 3, 1)$, $F = (2, 5, 1)$, $G = (3, 4, 2)$, *and* $H = (4, 5, 3)$. *The shaded region is a gap. In this example,* $LOAD = 4$, *but* $OPT = 5$.

By scaling, we will assume that each h_i is integral, in which case it is easy to see that without loss of generality $s(i)$ is integral too.

Dynamic storage allocation was proven NP-complete in 1976 by Stockmeyer. (See problem SR2 in Garey and Johnson [3].) The first polynomial-time, constant-factor approximation algorithm was given by Kierstead in 1988 [6], using a reduction from dynamic storage allocation to on-line coloring of interval graphs discovered by Woodall in 1973 [8] and independently by Chrobak and Slusarek in 1988 [1]. Kierstead's algorithm is an 80-approximation algorithm. Kierstead [7] later exhibited a 6-approximation algorithm for dynamic storage allocation. The next improvements were 5- and 3-approximation algorithms by Gergov [4, 5]. Neither of Gergov's two algorithms uses the Woodall–Chrobak–Slusarek reduction.

Define OPT to be the optimal makespan for a given instance. The *load* $L(t)$ *at (x-coordinate)* t is the sum of h_i over all jobs i that are live at t. The *maximum load* $LOAD$ (also L) is the maximum over all t of the load at t. $LOAD$ is a trivial lower bound on OPT. When all heights are equal, $OPT = LOAD$, and an optimal solution can be found via interval graph coloring. In general, however, OPT need not equal $LOAD$, as exemplified by Figure 1.1.

All four of the algorithms mentioned above actually find packings of makespan at most c times $LOAD$, not just c times OPT, for the respective c (80, 6, 5, or 3). We further study the relationship between OPT and $LOAD$ in dynamic storage allocation, providing new upper and lower bounds on the gap between them; all of our upper bounds are expressed algorithmically. Recall $L = LOAD$; denote the maximum job height by $h_{\max}$ and the minimum job height by $h_{\min}$. Our main results are as follows.

1. We devise an algorithm that yields makespan $(1 + O((h_{\max}/L)^{1/7}))L$. This bound is $L + o(L)$ when $h_{\max} = o(L)$. When $h_{\max}$ is bounded by a constant, we improve the makespan bound to $(1 + O(((\lg^2 L)/L)^{1/4}))L$. Note that the case $h_{\max} = o(L)$ does not subsume that of $h_{\max}$'s being a constant, because, e.g., L might itself be bounded by a constant. No result of the form $OPT \le L + o(L)$ is possible in the general case, since one can "scale up" the heights in the example of Figure 1.1 to get $L = 4k$ and $OPT = 5k$ for any positive integer k.

2. For any $\epsilon > 0$, there exists a $c > 0$ such that for all sufficiently large constants $h_{\max}$, there is a dynamic storage allocation instance with maximum job height $h_{\max}$, maximum load at most L, and $OPT \ge (1 + c(h_{\max}/L)^{1/2+\epsilon})L$, for infinitely many integers L. This is the first nontrivial lower bound to include the case when $h_{\max}$ is bounded by a constant. This lower bound shows that our upper bounds in (1) are

optimal up to the exact powers of $h_{\max}/L$. In particular, the "1/7" in the general upper bound (and the "1/4" in the bounded-$h_{\max}$ upper bound) cannot be replaced by any real greater than 1/2.

We also contribute the following corollaries to our main results.

1. For all $\epsilon > 0$, we give polynomial-time, $(2+\epsilon)$-approximation algorithms.
2. We give polynomial-time approximation schemes (PTASs) for the following cases:
   (a) $h_{\max} = o(L)$;
   (b) $h_{\max}$ is bounded by a constant.

Finally, using straightforward dynamic programming, we give a PTAS when $h_{\min} = \Omega(L \lg\lg n / \lg n)$ and a polynomial-time, exact solution when $L = O(\lg n / \lg\lg n)$.

We begin in section 2 by introducing our main tool: boxing jobs into larger rectangles so that the total wasted space in the rectangles remains small. Along with some results for simple cases (small load and large jobs) described in section 3, we use boxing to devise our algorithms for bounded-height jobs (section 4) and finally for the remaining cases (section 5). We present the lower bound in section 6.

2. Boxing jobs. Let Y be any set of jobs and t be any x-coordinate. $L_Y(t)$ denotes the load of the jobs in Y that are live at t.

To *box* a set Y of jobs means to place the jobs into a box b of minimum x-coordinate $x_b = \min\{x_j : j \in Y\}$, maximum x-coordinate $y_b = \max\{y_j : j \in Y\}$, and height $h_b \geq \sum_{j \in Y} h_j$. A *boxing* of a set Y into a set B of boxes is a partition of Y into at most $|B|$ subsets, each of which is then boxed into a distinct box $b \in B$. The boxes can be viewed as jobs in a modified instance. Then L_B (resp., $L_B(t)$) is well defined to mean the *load of the boxes* (resp., *at* t). In particular, any unused space in a box still counts toward the load contributed by that box. All of the boxing procedures that follow run in polynomial time.

2.1. Boxing for a fixed time.

LEMMA 2.1. *Given a set Y of unit-height jobs, all live at some fixed x-coordinate t, an integer* box-height parameter *H, and a sufficiently small positive ϵ, there exist a subset Y' of Y, $|Y - Y'| \leq 2H\lceil 1/\epsilon^2 \rceil$, a set B of boxes, each of height H, and a boxing of Y' into B such that at any x-coordinate u,*

$$L_B(u) \leq L_{Y'}(u) + 4\epsilon L_Y(u).$$

Proof. It is convenient to view a job starting at x and ending at y as a point (x, y) in the plane, as depicted in Figure 2.1(a)–(b). Now partition the jobs of Y into *strips*, as exemplified by Figure 2.1(c). The first two strips are defined as follows.

1. Create a vertical strip consisting of the $H\lceil 1/\epsilon^2 \rceil$ jobs with the earliest starting x-coordinates (or fewer if there are not enough jobs).

2. If any jobs remain, create a horizontal strip consisting of the $H\lceil 1/\epsilon^2 \rceil$ jobs that remain with the latest ending x-coordinates (or fewer if not enough jobs remain).

Define Y' to be the set of all jobs in neither the first vertical nor the first horizontal strip. Obviously $|Y - Y'| \leq 2H\lceil 1/\epsilon^2 \rceil$. Now partition the jobs of Y' into strips by repeating the following as long as jobs remain.

1. Create a vertical strip consisting of the $H\lceil 1/\epsilon \rceil$ jobs that remain with the earliest starting x-coordinates (or fewer if not enough jobs remain).

2. If any jobs remain, create a horizontal strip consisting of the $H\lceil 1/\epsilon \rceil$ jobs that remain with the latest ending x-coordinates (or fewer if not enough jobs remain).

Now for every vertical strip of Y', take the jobs in order of decreasing ending x-coordinate in groups of size H (the last group of the last strip possibly smaller),

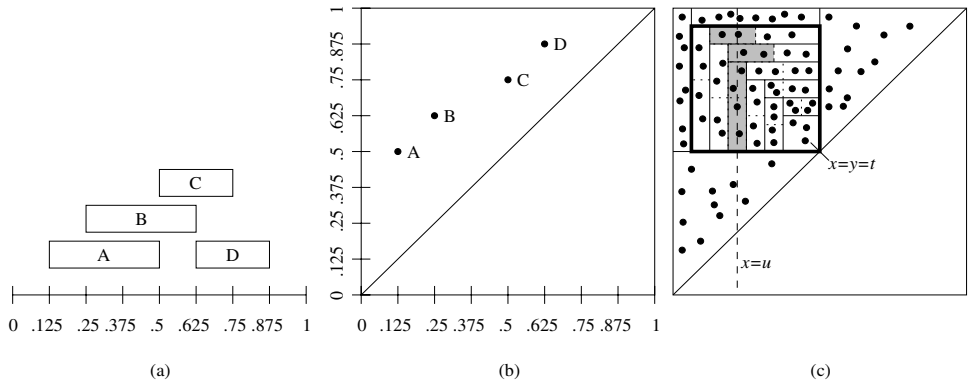


FIG. 2.1. (a) *Four jobs.* (b) *The jobs of* (a) *viewed as* (x, y) *points. Note that all jobs are above the* $x = y$ *diagonal.* (c) *Stripping a set of jobs with* $H = 2$ *and* $\epsilon = .5$. *The rectangle* R_t *defines the set* Y *of jobs. The first (leftmost) vertical and (topmost) horizontal strips each contains* $H\lceil 1/\epsilon^2 \rceil = 8$ *jobs. The remaining jobs, which comprise* Y', *are shown in the bold subrectangle.* Y' *is partitioned into strips containing* $H\lceil 1/\epsilon \rceil = 4$ *jobs each. Within each strip of* 4 *jobs in* Y', *the jobs are grouped (dotted lines) into groups of height* $H = 2$. *The groups that intersect the line* $x = u$ *are shaded.*

and box them. Similarly, for every horizontal strip of Y', take the jobs in order of increasing starting x-coordinate in groups of size H (the last group of the last strip possibly smaller), and box them.

We analyze the construction using Figure 2.1(c). For all reals u, the jobs live at u correspond to the rectangle $R_u = \{(x, y) : x < u < y\}$. The fact that the jobs in Y are all live at t means that all the corresponding vertices are inside the rectangle R_t.

A box (other than a last box) corresponds to a group of exactly H jobs. If all are live at u, then these jobs contribute exactly H to $L_{Y'}(u)$, because each job has unit height, and the box contributes exactly H to $L_B(u)$. If none of them is live at u, then they contribute 0 to $L_{Y'}(u)$, and the box contributes 0 to $L_B(u)$. The troublesome case is the one in which some but not all of the jobs are live at u, since the jobs may contribute as little as 1 to $L_{Y'}(u)$, while the box still contributes H to $L_B(u)$.

Assume without loss of generality that $u < t$. Then the troublesome case corresponds to groups of jobs whose vertices are on both sides of the line $x = u$. By construction, this can happen to at most one group in each horizontal strip of Y'. (Recall that points in a horizontal strip are grouped in order of increasing x-coordinates.) Moreover, this can happen to groups inside at most one vertical strip of Y'.

Notice that if the line $x = u$ intersects, say, k horizontal strips of Y', then the rectangle R_u entirely contains at least $k - 1$ vertical strips of Y'. (In fact, in the current case, R_u entirely contains at least k vertical strips of Y'. In the symmetric case, when $t < u$, the number is $k - 1$. We argue without loss of generality.)

If the rectangle R_u does not contain any job of Y', then the lemma trivially holds. Otherwise, R_u must contain all the jobs in the (only) vertical strip of $Y - Y'$, which number $H\lceil 1/\epsilon^2 \rceil$.

Observe that

$$L_B(u) - L_{Y'}(u) \leq kH + H\lceil 1/\epsilon \rceil + H, \tag{2.1}$$

where the first term accounts for the groups in the k horizontal strips that are intersected by $x = u$, the second term accounts for the groups in the single vertical strip

that is intersected by $x = u$, and the third term accounts for (possibly) the last group of the last strip.

But

$$L_Y(u) \geq H\lceil 1/\epsilon^2 \rceil + \max\{k-1, 0\} \cdot H\lceil 1/\epsilon \rceil, \tag{2.2}$$

where the first term accounts for the vertical strip of $Y - Y'$, and the second accounts for the $\max\{k-1, 0\}$ vertical strips of Y' that are contained in the rectangle R_u.

Now observe that

$$kH + H\lceil 1/\epsilon \rceil + H \leq (k-1)H + H/\epsilon + 3H; \tag{2.3}$$

$$H/\epsilon + 3H \leq H\epsilon(3/\epsilon + 1/\epsilon^2) \leq H\epsilon(4/\epsilon^2) \leq (4\epsilon)[H\lceil 1/\epsilon^2 \rceil]; \tag{2.4}$$

$$(k-1)H \leq \epsilon \max\{k-1, 0\} \cdot H\lceil 1/\epsilon \rceil \leq (4\epsilon)[\max\{k-1, 0\} \cdot H\lceil 1/\epsilon \rceil]. \tag{2.5}$$

Combining equations (2.1)–(2.5) yields

$$L_B(u) - L_{Y'}(u) \leq 4\epsilon L_Y(u). \qquad \square$$

We call the jobs in $Y - Y'$ *unresolved jobs*.

2.2. Boxing over all times. We bootstrap Lemma 2.1 so that we can box all the jobs (i.e., not just jobs live at a particular, fixed x-coordinate) with just a small amount of wasted space. Our main technical result is particularly interesting when $L \gg (H \lg H) \lg(1/\epsilon)/\epsilon^2$. We use this theorem in the following sections to devise approximation schemes for dynamic storage allocation.

THEOREM 2.2. *Given a set Z of jobs, each of height* 1*, an integer* box-height parameter *H, and a sufficiently small positive ϵ, there exist a set B of boxes, each of height H, and a boxing of Z into B such that for all x-coordinates t,*

$$L_B(t) \leq (1 + 4\epsilon)L_Z(t) + O\left(\frac{H \lg H}{\epsilon^2} \lg \frac{1}{\epsilon}\right).$$

To prove Theorem 2.2, we are going to apply Lemma 2.1 many times, boxing the unresolved jobs into additional boxes as we go along. Our general goal is to keep the wasted load (free space) in those additional boxes small at any x-coordinate. We use the following recursive method. Given are

1. a set X of jobs and an open *bounding interval* I, such that $\forall j \in X, I_j \subseteq I$;
2. a nonempty finite set of *critical x-coordinates* $T = \{\inf I = t_0 < t_1 < \cdots < t_q < t_{q+1} = \sup I\} \subseteq I \cup \{\inf I, \sup I\}$;
3. a set F of *free spaces*. Each free space is an open subinterval of I of height 1 *having endpoints in* T. Any free space $f \in F$ is called *spanning* if $f = I$ and *nonspanning* otherwise.

Initially, $X = Z$, $I = (0, 1)$, $T = \{0, t, 1\}$ for some arbitrary x-coordinate t at which some job from Z is live, and $F = \emptyset$. Recall that $I_j = (x_j, y_j)$ denotes the interval of job j. With the help of T, define partition

$$X = (R_1 \cup R_2 \cup \cdots \cup R_q) \cup (X_0 \cup X_1 \cup \cdots \cup X_q)$$

as follows. (See Figure 2.2.) First, define $X_i = \{j \in X : I_j \subseteq (t_i, t_{i+1})\}$ for $0 \leq i \leq q$. The X_i's contain precisely the jobs that are not live at any critical time.

Then define the R_i's recursively as in an interval tree [2]. Define $X' = X \setminus (X_0 \cup X_1 \cup \cdots \cup X_q)$, the set of jobs that are live at some (not necessarily unique) critical

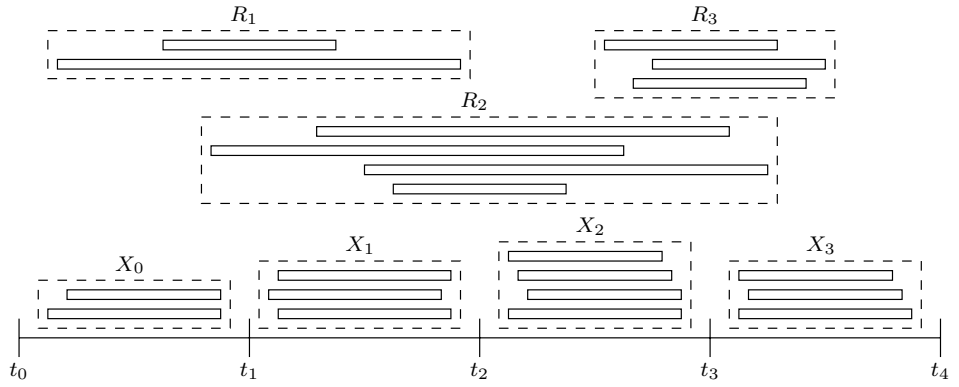


FIG. 2.2. *A set of jobs (solid rectangles) partitioned into subsets (dashed rectangles) X_i and R_j using critical x-coordinates $t_0, \ldots, t_4$.*

time. Note that $q \geq 1$. Define $R_{\lceil q/2 \rceil} = \{j \in X' : t_{\lceil q/2 \rceil} \in I_j\}$. Define P to be the set of remaining jobs j of X' with $y_j < t_{\lceil q/2 \rceil}$, and define Q to be the set of remaining jobs j of X' with $t_{\lceil q/2 \rceil} < x_j$. That is, $R_{\lceil q/2 \rceil}$ is the set of jobs that are live at $t_{\lceil q/2 \rceil}$; P is the set of jobs that are live at some critical time but end before $t_{\lceil q/2 \rceil}$; and Q is the set of jobs that are live at some critical time but start after $t_{\lceil q/2 \rceil}$. If $P \neq \emptyset$, recursively partition P using $\{t_1, t_2, \ldots, t_{\lceil q/2 \rceil - 1}\}$. Afterward, if $Q \neq \emptyset$, recursively partition Q using $\{t_{\lceil q/2 \rceil + 1}, t_{\lceil q/2 \rceil + 2}, \ldots, t_q\}$.

We will box the jobs in the R_i's, in the process establishing parallel, recursive instances of the decomposition to handle the X_i's. First, to each X_i associate a set F_i of intervals (free spaces), initially empty. As sections of free spaces in F are used to box jobs in the R_i's, the unused fragments will be deposited into the appropriate F_i's for use deeper in the recursion, to box jobs in the X_i's.

To box the jobs in the R_i's, first apply Lemma 2.1 to each R_i, $1 \leq i \leq q$, in any order; note that all jobs in R_i are live at t_i. For each i, this boxes all the jobs of R_i except for at most $2H\lceil 1/\epsilon^2 \rceil$ unresolved jobs. Now consider the set U of all the unresolved jobs from all the R_i's. Derive an optimal packing of U using interval graph coloring. (Recall that all jobs are of height one.) This packing has makespan L_U.

Let $s(F)$ denote the subset of spanning free spaces of F. If $|s(F)| < L_U$, create $\lceil (L_U - |s(F)|)/H \rceil$ boxes of height H and horizontal extent I. This yields $H\lceil (L_U - |s(F)|)/H \rceil$ new spanning free spaces; add them to F. Now there are at least as many spanning free spaces in F as rows of the packing of U.

For each $1 \leq j \leq L_U$, remove one spanning free space from F, and use it to place all the jobs in row j of the packing. This creates *gaps*, or unused portions, in the original free space, each of the form $[\alpha, \beta]$, where for some i, j: $t_i \leq \alpha < t_{i+1}$ and $t_j \leq \beta < t_{j+1}$; recall that $t_0 = \inf I$ and $t_{q+1} = \sup I$. For each such $[\alpha, \beta]$, if $i \neq j$, then split $[\alpha, \beta]$ into $(\alpha, t_{i+1}), (t_{i+1}, t_{i+2}), \ldots, (t_{j-1}, t_j), (t_j, \beta)$; and add (α, t_{i+1}) to F_i, (t_{i+1}, t_{i+2}) to F_{i+1}, ..., (t_{j-1}, t_j) to F_{j-1}, and (t_j, β) to F_j. Otherwise $(i = j)$, simply deposit (α, β) into F_i. This *fragments* the gaps.

Now all the jobs in all the R_i's are boxed. Consider the unused free spaces in F, if any. Each is of the form (t_i, t_j) for some $i \neq j$. Split each such (t_i, t_j) into $(t_i, t_{i+1}), (t_{i+1}, t_{i+2}), \ldots, (t_{j-1}, t_j)$. Add (t_i, t_{i+1}) to F_i, (t_{i+1}, t_{i+2}) to F_{i+1}, ..., and (t_{j-1}, t_j) to F_{j-1}. This *passes down* the unused free spaces to the subproblems.

In parallel for each $\ell = 0, 1, 2, \ldots, q$, if $X_\ell \neq \emptyset$, recursively apply the construction

with new $X \leftarrow X_\ell$, new free space set $F \leftarrow F_\ell$, new bounding interval $I \leftarrow (t_\ell, t_{\ell+1})$, and new critical x-coordinate set $T \leftarrow \{\text{endpoints of elements of } F_\ell\} \cup \{t_\ell, t_{\ell+1}\}$. Note that this preserves the invariant that the endpoints of intervals in F are in T for each recursive subproblem. Also note that the unused free spaces passed down span their entire respective boundary intervals, whereas the gaps that were fragmented need not do so. This completes the construction.

2.3. Analysis. Fix a real u. (Recall that to prove Theorem 2.2 we want to bound the load of the boxes at any x-coordinate u in terms of the load of the original jobs at u.) For any m, define $I_m(u)$ to be the bounding interval of the depth-m recursive call whose bounding interval contains x-coordinate u. Similarly, define $T_m(u)$ to be the set of critical x-coordinates, $p_m(u) = |T_m(u)|$, and $U_m(u)$ to be the set of unresolved jobs from all the R_i's in that same depth-m recursive call. Define $F_m(u)$ to be the set of free spaces during the same depth-m recursive call, and note that this set changes during that procedure. Figure 2.2, for example, depicts for some m a piece of the depth-m stage of the recursion; i.e., $I_m(u) = (t_0, t_4)$ for $u \in (t_0, t_4)$. The jobs in each X_i will form the set of jobs for each subproblem at depth $m+1$; i.e., $I_{m+1}(u) = (t_0, t_1)$ for $u \in (t_0, t_1)$, $I_{m+1}(u) = (t_1, t_2)$ for $u \in (t_1, t_2)$, etc.

LEMMA 2.3. *For all m, $p_{m+1}(u) \le 2H\lceil 1/\epsilon^2 \rceil \lceil \lg(p_m(u)+1) \rceil + 2$.*

Proof. Note that $p_0(u) = 3$. In the depth-m recursive call, the definition of the R_i's implies that the number of R_i's containing jobs with endpoints in $I_{m+1}(u)$ is at most $\lceil \lg(p_m(u)+1) \rceil$. (The process of repeatedly looking at the middle index resembles binary search, and the worst-case running time of binary search on a list of length p is $\lceil \lg(p+1) \rceil$.) For each such R_i, Lemma 2.1 is applied, and each application yields at most $2H\lceil 1/\epsilon^2 \rceil$ unresolved jobs. Each unresolved job contributes at most one new endpoint and hence at most one critical x-coordinate to each of two different intervals $I_{m+1}(\cdot)$. The total number of critical x-coordinates contributed to $I_{m+1}(u)$ is thus at most $2H\lceil 1/\epsilon^2 \rceil \lceil \lg(p_m(u)+1) \rceil + 2$, where the "+2" comes from the endpoints $\inf I_{m+1}(u)$ and $\sup I_{m+1}(u)$. □

COROLLARY 2.4. *For all m, $p_m(u) = O(\frac{H \lg H}{\epsilon^2} \lg \frac{1}{\epsilon})$.*

Proof sketch. If $a_{m+1} \le k \lg a_m$ for all m, and $a_0 \le 3k \lg k$, then $a_m \le 3k \lg k$ for all m, a fact that is easily proven by induction on m. □

Let P denote the upper bound of the corollary. For all m, the number of critical x-coordinates in the subproblem defined by bounding interval $I_m(u)$ is at most P. Let $f_m(u)$ denote the number of free spaces in $F_m(u)$ *at the beginning* of that recursive call, and let $f'_m(u)$ denote the number of free spaces in $F_m(u)$ *at the end* of that recursive call. Abusing notation, let $L_m(u)$ denote the load of $U_m(u)$, the set of unresolved jobs from all the R_i's in the depth-m recursive call over bounding interval $I_m(u)$.

LEMMA 2.5. *For all m, $L_m(u) \le 2H\lceil 1/\epsilon^2 \rceil \lceil \lg(P+1) \rceil$.*

Proof. As above, at most $\lceil \lg(P+1) \rceil$ R_i's include jobs that contain u' for any $u' \in I_m(u)$. By Lemma 2.1 each such R_i contributes at most $2H\lceil 1/\epsilon^2 \rceil$ unresolved jobs to $U_m(u)$. □

COROLLARY 2.6. *For all m, $f'_m(u) \le f_m(u) + 2H\lceil 1/\epsilon^2 \rceil \lceil \lg(P+1) \rceil$.*

Proof. Boxing the unresolved jobs in the depth-m recursive call requires creating at most $\lceil L_m(u)/H \rceil$ new boxes of height H, which adds at most $H\lceil L_m(u)/H \rceil$ new (spanning) free spaces to $F_m(u)$, after which free spaces are only removed from $F_m(u)$. Lemma 2.5 completes the proof. □

LEMMA 2.7. *For all m,*

1. $f_{m+1}(u) \le f_m(u) + 2H\lceil 1/\epsilon^2 \rceil \lceil \lg(P+1) \rceil$;
2. *if* $f_m(u) \ge 4H\lceil 1/\epsilon^2 \rceil \lceil \lg(P+1) \rceil$, *then* $f_{m+1}(u) \le f_m(u)$.

Proof. 1. This follows from Corollary 2.6 and the fact that the fragmented gaps from any used spanning free space of $F_m(u)$ are distributed to distinct subproblems.

2. The free spaces of $F_m(u)$ at the beginning of the depth-m recursive call come from two sources.

(i) Unused free spaces of $F_{m-1}(u)$ that were passed down; as noted above, they yield spanning free spaces of $F_m(u)$, i.e., free spaces that span the entire interval $I_m(u)$.

(ii) Fragmented free spaces obtained during the depth-$(m-1)$ recursive call by placing an unresolved job in a free space of $F_{m-1}(u)$. Lemma 2.5 implies that there are at most $2H\lceil 1/\epsilon^2\rceil\lceil \lg(P+1)\rceil$ such spaces.

Assume $f_m(u) \geq 4H\lceil 1/\epsilon^2\rceil\lceil \lg(P+1)\rceil$. The number of spanning free spaces is the total number of free spaces minus the number of nonspanning free spaces. Because there are at most $2H\lceil 1/\epsilon^2\rceil\lceil \lg(P+1)\rceil$ nonspanning free spaces, $F_m(u)$ starts with at least

$$4H\lceil 1/\epsilon^2\rceil\lceil \lg(P+1)\rceil - 2H\lceil 1/\epsilon^2\rceil\lceil \lg(P+1)\rceil = 2H\lceil 1/\epsilon^2\rceil\lceil \lg(P+1)\rceil$$

spanning free spaces. When the unresolved jobs are boxed during the depth-m recursive call, there are enough spanning free spaces to fit *all* the unresolved jobs (again by Lemma 2.5), so no new free spaces are created. Thus, $f_{m+1}(u) \leq f_m(u)$. □

COROLLARY 2.8. *For all m, $f_m(u) = O\big(\frac{H \lg H}{\epsilon^2} \lg \frac{1}{\epsilon}\big)$.*

Proof. The proof follows from Corollary 2.4 and Lemma 2.7. □

COROLLARY 2.9. *For all m, $f'_m(u) = O\big(\frac{H \lg H}{\epsilon^2} \lg \frac{1}{\epsilon}\big)$.*

Proof. The proof follows from Corollaries 2.4, 2.6, and 2.8. □

We can now finish the proof of Theorem 2.2. Consider any x-coordinate u, and let m' denote the greatest depth of the recursion for which there existed an $I_{m'}(u)$. Of the jobs in Z that are live at u, let Z_R denote those that are resolved during applications of Lemma 2.1, and let Z_U denote the rest. The load at u of the boxes used to resolve jobs in Z_R is at most $L_{Z_R}(u) + 4\epsilon L_Z(u)$, by Lemma 2.1. The load at u of the boxes used to place jobs in Z_U is $L_{Z_U}(u) + f'_{m'}(u)$, because jobs in Z_U are only placed into free spaces in the various $F_m(u)$'s, and unused free spaces containing u in $F_m(u)$ for any $0 \leq m < m'$ are inherited by $F_{m+1}(u)$. Therefore, using Corollary 2.9 and the fact that $L_Z(u) = L_{Z_R}(u) + L_{Z_U}(u)$, the total storage allocated at x-coordinate u does not exceed

$$\big[L_{Z_R}(u) + 4\epsilon L_Z(u)\big] + \big[L_{Z_U}(u) + f'_{m'}(u)\big] = (1+4\epsilon)L_Z(u) + O\left(\frac{H \lg H}{\epsilon^2} \lg \frac{1}{\epsilon}\right). \quad \square$$

3. Dynamic programming solutions. In addition to boxing, our later results use the following results based on dynamic programming to solve simple cases.

THEOREM 3.1. *The optimal makespan can be determined in $O(\mathrm{poly}(n)(3L)^{2L+1})$ time.*

Proof. First, using the fact that $OPT \leq 3L$ [5], guess the optimal makespan M^*. Build an array T with, for each x, an entry for each feasible placement C_x of jobs that cross the vertical line at x. Now define $T[C_x]$ to be 0–1, with $T[C_x] = 1$ if and only if the set of jobs that intersect $[0, x)$ can be placed using height at most M^* with a placement that respects C_x. We have $T[C_x] = 1$ if and only if there is a $C_{x-1} = 1$ such that C_{x-1} is compatible with C_x and $T[C_{x-1}] = 1$.

There are $3L$ possibilities for M^*. There are $2n$ possibilities for x. For each x, there are $(M^*)^L \leq (3L)^L$ configurations C_x. Computing $T[C_x]$ takes time at most $(3L)^L$. The overall time bound is thus

$$O((3L)(3L)^L(3L)^L\mathrm{poly}(n)),$$

which is $O(\text{poly}(n)(3L)^{2L+1})$. □

COROLLARY 3.2. *Dynamic storage allocation can be solved optimally in polynomial time when* $L = O(\lg n / \lg\lg n)$.

Proof. The proof follows immediately from Theorem 3.1. □

THEOREM 3.3. *Let* C *be a positive real and let* $\alpha = \alpha(n)$ *be a positive function of* n *such that for all sufficiently large* n, $\alpha(n) \geq \lg\lg n/(C \lg n)$. *There is a PTAS for the special case of dynamic storage allocation in which* $h_{\min}/L \geq \alpha(n)$.

Proof. Given C, α, and $\epsilon > 0$, apply the dynamic program in the proof of Theorem 3.1 to the same jobs, except with height h replaced by $\lceil h/(\epsilon\alpha(n)L)\rceil$. Let L' be the load in the new problem. If there were no ceiling in the definition of the new heights, the load in the new problem would be $L/(\epsilon\alpha(n)L) = 1/(\epsilon\alpha(n))$. Since each original height h satisfies $h/(\epsilon\alpha(n)L) \geq 1/\epsilon$, the ceiling introduces an additional factor of at most $1+\epsilon$. It follows that $L' \leq (1+\epsilon)/(\epsilon\alpha(n)) \leq (2C/\epsilon)\lg n/\lg\lg n$ if n is large enough and $\epsilon \leq 1$. The new problem can be solved exactly in polynomial time by Corollary 3.2. □

4. Algorithm for bounded-height jobs. Let X be the set of all jobs and ϵ be a sufficiently small positive real. Recall that L is a trivial lower bound to OPT. If all the jobs have the same height, then the problem reduces to interval graph coloring and can be solved optimally in a greedy fashion; furthermore, $OPT = L$. Our algorithm will be a reduction to this simple case. Let $H = h_{\max}\lceil 1/\epsilon\rceil$, and assume that the maximum height of a job, $h_{\max}$, is a constant.

ALGORITHM.

1. If $L \leq C\frac{1}{\epsilon^4}\lg^2\frac{1}{\epsilon}$, for some C to be determined later, apply Corollary 3.2 and halt.

2. For each $h = 1, 2, 3, \ldots, h_{\max}$:

(i) Let X_h denote the set of jobs of height h. Let $H_h = \lfloor H/h\rfloor h$.

(ii) Scale each job in X_h down by a factor of h, apply Theorem 2.2 with box-height parameter $\lfloor H/h\rfloor$ and the given ϵ to this new set of jobs, and then scale the jobs back up by the same factor h.

3. Enlarge the boxes so that they all have height exactly H. Call the new set of boxes B'.

4. Apply the greedy algorithm for interval graph coloring to B'.

THEOREM 4.1. *The makespan of the packing produced by the algorithm is at most* $(1+15\epsilon)OPT$. *(Recall that* $h_{\max}$ *is a constant and* ϵ *is sufficiently small.)*

Proof. Assume for now that the algorithm does not stop in step 1. We argue for all t. By Theorem 2.2, step 2 produces a boxing B_h of the jobs of X_h into boxes of height H_h such that

$$L_{B_h}(t) \leq (1+4\epsilon)L_{X_h}(t) + O\left(\frac{1}{\epsilon^2}\cdot\frac{H}{h}\left(\lg\frac{H}{h}\right)\lg\frac{1}{\epsilon}\right)h. \tag{4.1}$$

(The final h comes from the scaling back up at the end of step (2); we have applied the theorem to jobs of height 1 with box-height parameter there equal to $\lfloor H/h\rfloor = H_h/h$.) Let $B = B_1 \cup B_2 \cup \cdots \cup B_{h_{\max}}$. Adding (4.1) yields

$$L_B(t) \leq (1+4\epsilon)L_X(t) + O\left(h_{\max}\frac{H\lg H}{\epsilon^2}\lg\frac{1}{\epsilon}\right).$$

Enlarging the boxes increases them by at most a factor of $1/(1-\epsilon)$. All together,

$$\begin{aligned}\text{makespan} &= \max_t L_{B'}(t)\\ &\le \max_t \left[\frac{1}{1-\epsilon}\left((1+4\epsilon)L_X(t) + O\left(h_{\max}\frac{H\lg H}{\epsilon^2}\lg\frac{1}{\epsilon}\right)\right)\right]\\ &\le \frac{1+4\epsilon}{1-\epsilon}\max_t L_X(t) + \frac{1}{1-\epsilon}O\left(h_{\max}\frac{H\lg H}{\epsilon^2}\lg\frac{1}{\epsilon}\right).\end{aligned}$$

$$\begin{aligned}\text{Thus, makespan} &\le \frac{1+4\epsilon}{1-\epsilon}L + O\left(h_{\max}\frac{H\lg H}{\epsilon^2}\lg\frac{1}{\epsilon}\right)\\ &\le \frac{1+4\epsilon}{1-\epsilon}OPT + O\left(h_{\max}\frac{H\lg H}{\epsilon^2}\lg\frac{1}{\epsilon}\right).\end{aligned} \tag{4.2}$$

Choose C so that the error term $O\big(h_{\max}\frac{H\lg H}{\epsilon^2}\lg\frac{1}{\epsilon}\big) \le \epsilon L$, which is possible under the assumptions that $L > C\frac{1}{\epsilon^4}\lg^2\frac{1}{\epsilon}$, $h_{\max}$ is a constant, and $H = h_{\max}\lceil 1/\epsilon\rceil$. If $L \le C\frac{1}{\epsilon^4}\lg^2\frac{1}{\epsilon}$, then the algorithm stops in step (1), and Theorem 3.1 applies. Otherwise, it follows that makespan $\le (1+15\epsilon)OPT$, because $L \le OPT$. ∎

THEOREM 4.2. *The makespan produced by the packing is* $(1+O(((\lg^2 L)/L)^{1/4}))L$. *(Recall that* $h_{\max}$ *is a constant.)*

Proof. Set $\epsilon = \sqrt{\lg L}/L^{1/4}$, and run the algorithm starting in step (2). The claim follows from inequality (4.2). ∎

5. Algorithms for larger jobs. From Theorem 2.2, we can prove the following corollary.

COROLLARY 5.1. *Let* H *be a positive integer* box-height parameter, $h_{\min}$ *be a positive real, and* $\epsilon > 0$ *be a sufficiently small* error parameter. *Given a set* Z *of jobs, each of height between* $h_{\min}$ *and* ϵH, *there exist a set* B *of boxes, each of height* H, *and a boxing of* Z *into* B *such that for all* x*-coordinates* t,

$$L_B(t) \le (1+9\epsilon)L_Z(t) + O\left(\frac{H\lg^2(H/h_{\min})}{\epsilon^4}\right).$$

Proof. We construct an appropriate boxing. First, round the job heights: each height h is rounded up to $\lfloor(1+\epsilon)^i\rfloor$, where i is defined by $(1+\epsilon)^{i-1} < h \le (1+\epsilon)^i$. Let Y denote the resulting set of rounded jobs.

Now partition the jobs according to their heights. For each rounded height h, let Y_h denote the set of jobs of height h. Divide the heights of all jobs in Y_h by h; apply Theorem 2.2 with box-height parameter $\lfloor H/h\rfloor$; and then multiply all box heights by h to get a set B_h of boxes of height at most H. The output is a set $B = \bigcup_h B_h$ of boxes, which we can assume are all of height H.

Let us analyze this construction. There are approximately $\log_{(1+\epsilon)}(\epsilon H/h_{\min}) = O(\lg(H/h_{\min})/\epsilon)$ parts in the partition. Applying Theorem 2.2 to Y_h yields

$$\begin{aligned}\forall t,\quad L_{B_h}(t) &\le (1+4\epsilon)L_{Y_h}(t) + O\left(h\frac{\lfloor H/h\rfloor\lg\lfloor H/h\rfloor}{\epsilon^2}\lg\frac{1}{\epsilon}\right)\\ &\le (1+4\epsilon)L_{Y_h}(t) + O\left(\frac{H\lg(H/h_{\min})}{\epsilon^3}\right),\end{aligned}$$

because $h_{\min} \le h$ and $\lg(1/\epsilon) \le 1/\epsilon$. Summing over h, we get

$$\forall t,\quad L_B(t) \le (1+4\epsilon)L_Y(t) + O\left(\frac{H\lg^2(H/h_{\min})}{\epsilon^4}\right).$$

Since rounding increased heights by a factor of $1+\epsilon$ at most, we have $L_Y(t) \le (1+\epsilon)L_Z(t)$. Since $(1+\epsilon)(1+4\epsilon) \le 1+9\epsilon$ for $\epsilon \le 1$, the corollary follows. □

We are now ready to prove our main theorem.

THEOREM 5.2. *For any* fixed $\epsilon \in (0,1]$ *there exists a polynomial-time algorithm, which depends on* ϵ*, that takes an arbitrary set* X *of jobs as input and produces a feasible solution to dynamic storage allocation on* X *with makespan at most* $(1+c\epsilon)L + O(h_{\max}/\epsilon^6)$*, where* c *is a universal constant.*

Proof. It suffices to prove the theorem for $\epsilon \le \epsilon_0$ for some small, positive ϵ_0, for the following reason. Suppose for all $\epsilon \le \epsilon_0$ we get a makespan bound of $(1+c_0\epsilon)L + c_1(h_{\max}/\epsilon^6)$ for some constants c_0 and c_1. Then for $\epsilon \in (\epsilon_0, 1]$ we simply use $(1+c_0\epsilon_0)L + c_1 h_{\max}/\epsilon_0^6 \le (1+c_0\epsilon)L + (c_1/\epsilon_0^6)h_{\max}/\epsilon^6$. Similarly, by reducing ϵ by a factor of at most two, we may assume that $1/\epsilon \in \mathbb{Z}$.

We are going to apply Corollary 5.1 repeatedly, boxing the smallest jobs so as to increase the minimum job height $h_{\min}$ until it gets close enough to the maximum job height $h_{\max}$ that we can finish with a last application of Corollary 5.1.

We argue for all t. Let r denote the ratio $h_{\max}/h_{\min}$, and recall that $L = L_X$. Assume first that $\lg^2 r \ge 1/\epsilon$, and set $\mu = \epsilon/\lg^2 r$ and $H = \lceil \mu^5 h_{\max}/\lg^2 r \rceil$. Consider the partition $X = X_s \cup X_\ell$, where X_s denotes the jobs of height at most μH and $X_\ell = X \setminus X_s$. Now apply Corollary 5.1 to X_s with box-height parameter H and error parameter μ. This yields a set B_s of boxes of height H into which the jobs of X_s fit such that for some constants c_2 and c_3,

$$\begin{aligned} L_{B_s}(t) &\le (1+9\mu)L_{X_s}(t) + O\left(\frac{H\lg^2(H/h_{\min})}{\mu^4}\right) \\ &\le (1+9\mu)L_{X_s}(t) + c_2\mu h_{\max} \\ &\le L_{X_s}(t) + c_3\mu L(t) \\ &= L_{X_s}(t) + (c_3\epsilon/\lg^2 r)L(t). \end{aligned}$$

Now consider B_s as a set of jobs and the revised problem on $X' = B_s \cup X_\ell$. For this new problem, the load $L'(t)$ is at most $(1 + c_3\epsilon/\lg^2 r)L(t)$. Moreover, the new minimum height $h'_{\min}$ is at least μH, and the maximum height remains at most $h_{\max}$, so we get the new ratio

$$r' \le \frac{h_{\max}}{h'_{min}} \le \frac{h_{\max}}{\mu\lceil \mu^5 h_{\max}/\lg^2 r\rceil} \le \frac{h_{\max}}{\mu^6 h_{\max}/\lg^2 r} = \frac{\lg^2 r}{\mu^6} = \frac{\lg^{14} r}{\epsilon^6} \le \lg^{26} r.$$

Recall that the above construction was conditioned on $\lg^2 r \ge 1/\epsilon$. For ϵ sufficiently small, this implies $\lg^{26} r \le \sqrt{r}$ and hence that $r' \le \sqrt{r}$ and also $\lg^2 r' \le \lg^2 \sqrt{r} \le (1/4)\lg^2 r$.

Iterate the above boxing of small jobs, each time using new error parameter $\mu' = \epsilon/\lg^2 r'$, until it yields a problem X^* with minimum job height $h^*_{\min}$ for which the ratio $r^* = h_{\max}/h^*_{\min}$ is such that $\lg^2 r^* < 1/\epsilon$. Since the ratio decreases by at least a square root each time, this process terminates in polynomial time.

Let $(r_0, \ldots, r_p = r^*)$ be the sequence of ratios, and let $(L_0(t), \ldots, L_p(t) = L^*(t))$ be the sequence of loads. For $0 \le i < p$ we know that

$$L_{i+1}(t) \le (1 + c_3\epsilon/\lg^2 r_i)L_i(t); \tag{5.1}$$

$$\lg^2 r_{i+1} \le (1/4)\lg^2 r_i. \tag{5.2}$$

For some constant c_4, we therefore get

$$\begin{aligned}
L^*(t) &\le L_0(t) \prod_{i=0}^{p-1} (1 + c_3\epsilon/\lg^2 r_i) \quad \text{by (5.1)} \\
&\le L_0(t) \exp\left(\sum_{i=0}^{p-1} (c_3\epsilon/\lg^2 r_i)\right) \\
&\le L_0(t) \exp(c_4\epsilon/\lg^2 r_{p-1}) \quad \text{by (5.2)} \\
&\le L_0(t) \exp(c_4\epsilon^2) \quad \text{because } \lg^2 r_{p-1} \ge 1/\epsilon \\
&\le (1 + 2c_4\epsilon^2) L_0(t) \quad \text{for } \epsilon \text{ sufficiently small.}
\end{aligned}$$

Now apply Corollary 5.1 to all of X^* with box-height parameter $H = h_{\max}/\epsilon$ (recall that $1/\epsilon \in \mathbb{Z}$) and error parameter ϵ; this is the "last application" of Corollary 5.1 to which we alluded earlier. For some constant c, this yields a final set X^f of jobs of identical height H and with load

$$\begin{aligned}
L^f(t) &= (1 + 9\epsilon) L^*(t) + O\left(\frac{H \lg^2(H/h^*_{\min})}{\epsilon^4}\right) \\
&\le (1 + 9\epsilon)(1 + 2c_4\epsilon^2) L_0(t) + O\left(\frac{H \lg^2(H/h^*_{\min})}{\epsilon^4}\right) \\
&\le (1 + c\epsilon) L_0(t) + O\left(\frac{h_{\max} \lg^2(r^*/\epsilon)}{\epsilon^5}\right).
\end{aligned}$$

Noting that $\lg^2 r^* < 1/\epsilon$ and $\lg^2(1/\epsilon) < 1/\epsilon$ for ϵ sufficiently small, we derive $\lg^2(r^*/\epsilon) \le 4/\epsilon$. Therefore

$$L^f(t) \le (1 + c\epsilon) L_0(t) + O(h_{\max}/\epsilon^6). \qquad \square$$

COROLLARY 5.3. *There exists a polynomial-time algorithm that takes an arbitrary set X of jobs as input and produces a feasible solution to dynamic storage allocation on X with makespan at most $(1 + O((h_{\max}/L)^{1/7}))L$.*

Proof. Apply Theorem 5.2 to X with $\epsilon = (h_{\max}/L)^{1/7}$. $\square$

COROLLARY 5.4. *Fix some function $f(m) = o(m)$. Then there is a PTAS for dynamic storage allocation when $h_{\max} \le f(L)$.*

Proof. Given $\epsilon > 0$, there is an L_0 such that $f(L) \le \epsilon^7 L$ for all $L \ge L_0$. If $L < L_0$, use the dynamic program of Theorem 3.1. If $L \ge L_0$, then

$$h_{\max} \le f(L) \le \epsilon^7 L.$$

Apply the algorithm of Theorem 5.2. The error term of Theorem 5.2 is $O(h_{\max}/\epsilon^6) \le c'\epsilon L$ for some constant c'. Hence the makespan of the schedule is at most $(1 + (c + c')\epsilon)L$. $\square$

THEOREM 5.5. *For all $\epsilon > 0$, there exists a polynomial-time $(2+\epsilon)$-approximation algorithm for dynamic storage allocation.*

Proof. Consider some small positive δ to be determined later. Let $X = X_s \cup X_\ell$, where X_s is the set of jobs of height less than $\delta^7 L$ and $X_\ell = X \setminus X_s$. Use Theorem 5.2 with error parameter δ to pack the jobs in X_s, yielding a $(1 + c'\delta)$-approximation for some constant c'. Apply the $(1+\delta)$-approximation algorithm implied by Theorem 3.3 with the same δ to pack the jobs in X_ℓ, which is possible because the load divided by the minimum height is at most $1/\delta^7$, which is certainly at most $C \lg n / \lg\lg n$ for $C = 1/\delta^7$; this yields a $(1 + \delta)$-approximation. Choose δ so that $\delta(c' + 1) = \epsilon$. $\square$

6. Lower bounds for *OPT − LOAD*. Fix a positive integer d. Given a bipartite multigraph $G = (X, Y, E)$, $X = Y = \{1, 2, \ldots, n\}$, build an instance of dynamic storage allocation as follows. Start with n rectangles of height d, with $x_i = 0$, $y_i = i$ for $i = 1, 2, \ldots, n$, and then add n more rectangles of height d, with $x_i = 3n - i$, $y_i = 3n$ for $i = 1, 2, \ldots, n$. Call these $2n$ items *d-items*. For each $e = (j, k) \in E$ (with multiplicity, as E is a multiset), $1 \le j, k \le n$, add one item of height 1 having $x_i = j$, $y_i = 3n - k$. Call these jobs *1-items*. These are all the jobs. Call the instance I.

LEMMA 6.1. *If G is d-regular, then $L_I(t) \le dn$ for all $t \in (0, 3n)$.*

Proof. For any noninteger $t \in [n, 2n]$, the statement of the lemma is obvious, since the set of jobs that intersect $(n, 2n)$ is the set of all 1-items, that set has size dn, and each such job has height 1.

For any noninteger $t \in [0, n]$, let $j = \lfloor t \rfloor$. Crossing t are $n - j$ d-items (these jobs have $x_i = 0$) and dj 1-items (these have $x_i \in \{1, 2, \ldots, j\}$), because G is d-regular, and no others. Hence the load at t is $(n - j) \cdot d + (dj) \cdot 1 = dn$.

For a noninteger $t \in [2n, 3n]$, the argument is symmetric to the case of $t \in [0, n]$.

For any integer $t \in (0, 3n)$, $L_I(\cdot)$ achieves a local minimum at t. ∎

LEMMA 6.2. *For any positive integers n, d, f, and $s < \frac{n^{1/2-1/f}}{\sqrt{d}}$, there exists a d-regular bipartite multigraph (X, Y, E) with $X = Y = \{1, \ldots, n\}$ such that for any subsets X' of s consecutive vertices from X and Y' of s consecutive vertices from Y, the number of edges induced by $X' \cup Y'$ is smaller than f.*

Proof. Consider the following probability space of n-vertex-by-n-vertex d-regular bipartite multigraphs. Choose d independent permutations π of $\{1, \ldots, n\}$; for each, add edges between $i \in X$ and $\pi(i) \in Y$ for $1 \le i \le n$. We will show that the probability that a graph not of the claimed structure exists is less than 1; the claim then follows. Consider such a graph $G = (X, Y, E)$. There are some j and k such that $X' = \{j, \ldots, j + s - 1\} \subseteq X$, $Y' = \{k, \ldots, k + s - 1\} \subseteq Y$, and the subgraph induced by $X' \cup Y'$ has at least f edges. Consider any f such edges and fix an arbitrary order of them; this forms a sequence of edges $F = ((u_1, v_1), \ldots, (u_f, v_f))$, where each $u_i \in X'$ and each $v_i \in Y'$. There are no more than n^2 choices for the pair (j, k) and no more than s^{2f} possible sequences F. The probability that any particular edge exists is at most d/n. By the union bound, the probability that G exists is therefore no more than $P = n^2 s^{2f} (d/n)^f$. Simple algebra yields $P < 1$ when $s < \frac{n^{1/2-1/f}}{\sqrt{d}}$. ∎

THEOREM 6.3. *For all positive integers n and d, there is an instance of dynamic storage allocation with maximum job height d, $L(t) \le dn$ for all t, and $OPT - LOAD \ge \frac{n^{1/2-1/\lceil d/2 \rceil}}{12\sqrt{d}}$.*

Proof. Let $G = (X, Y, E)$ be a d-regular bipartite multigraph such that $|X| = |Y| = n$. Build the instance I associated with G, as above, with $2n$ d-items and dn 1-items. Lemma 6.1 shows that the load is at most dn everywhere.

Assume an optimal solution, and denote by Z the region defined by the isothetic bounding box of the d-items. Z is the rectangle $[0, 3n] \times [\min, \max]$, where min is the minimum y-coordinate of the bottoms of the $2n$ d-items and max is the maximum y-coordinate of the tops of the $2n$ d-items. Define Z' to be the set of 1-items placed outside Z. Because $\max - \min \ge dn$ and the load is at most dn everywhere, it follows that the load of Z' lower bounds $OPT - LOAD$ and that the area of the jobs in Z' lower bounds the empty space inside Z.

Now, for any positive real $s < n$, assume that fewer than $s/12$ 1-items are placed outside Z and also that $\max - \min < dn + s/12$. The area of Z is thus $(\max - \min)(3n) < (dn + s/12)(3n)$. The total area of all the jobs is $(dn)(3n)$. There-

fore, if all the jobs were placed in Z, the empty space in Z would be at most $(s/12)(3n)$. For each job placed outside Z, the empty space inside Z grows by at most $3n$. The total empty space inside Z is thus less than $2(s/12)(3n) = sn/2$.

Consider a d-item to be comprised of d unit-height *rows*, which are placed contiguously in the plane. Define the *left* (resp., *right*) *d-items* to be those whose left (resp., right) endpoints are 0 (resp., $3n$). Call region $(3n-1, 3n) \times (k, k+1)$ for any integer $k \in [\min, \max - 1]$ a *right gap* if it is unoccupied by every right d-item. Clearly $OPT - LOAD$ is at least the number of right gaps, so we may assume this number is less than $n/2$. It follows that at least $n/2$ left d-items are adjacent to no right gaps; i.e., for each such left d-item x, there is a row of some right d-item to the right of each one of x's d rows. Call these the *good left d-items.*

For any left d-item J, define its *neighbor right d-items* to be the (at most two) right d-items whose y-coordinates overlap with J, and call the actual rows of those neighbors that occupy the same y-coordinates as J its *neighbor right rows.* Because there are at least $n/2$ good left d-items and the total empty space inside Z is less than $sn/2$, there must be at least one good left d-item J with less than s empty space between it and its neighbor right rows. Because $s < n$, there is some 1-item x placed between each row of J and the corresponding neighbor right row; furthermore, the left endpoint of x is within $s-1$ of the right endpoint of J, and the right endpoint of x is within $s-1$ of the left endpoint of the corresponding neighbor right row. For one of the at most two neighbor right d-items K of J, there are at least $\lceil d/2 \rceil$ neighbor right rows of J belonging to K. In G, therefore, there are subsets of $\lfloor s \rfloor$ consecutive vertices of X (starting at vertex $j \in X$, where j is the right endpoint of J) and $\lfloor s \rfloor$ consecutive vertices of Y (starting at vertex $k - \lfloor s \rfloor + 1 \in Y$, where k is the left endpoint of K) with at least $\lceil d/2 \rceil$ edges between them, corresponding to these 1-items.

If $s < \frac{n^{1/2-1/\lceil d/2 \rceil}}{\sqrt{d}}$, Lemma 6.2 (with $f = \lceil d/2 \rceil$) shows that there is some G' without such a window of edges. By contradiction, therefore, for any such G' there must be at least $s/12$ 1-items placed outside Z, or else $\max - \min \geq s/12$; in either case, $OPT - LOAD \geq s/12$. The theorem follows by choosing $s = \frac{n^{1/2-1/\lceil d/2 \rceil}}{\sqrt{d}}$ and noting that in fact $OPT - LOAD \geq \lceil s/12 \rceil$. □

COROLLARY 6.4. *For any $\epsilon > 0$, there exist $c, d > 0$ such that for all sufficiently large integers L, there is an instance of* dynamic storage allocation *with maximum job height d, $L(t) \leq L$ for all t, and $OPT - LOAD \geq cL^{1/2-\epsilon}$.*

Proof. Fix $d = \lceil 2/\epsilon \rceil$, and choose a large $L \in \mathbb{Z}$. Define $n, r \in \mathbb{Z}$ such that $n > 0$, $0 \leq r < d$, and $L = nd + r$. Define $L' = nd$. By Theorem 6.3, there is a dynamic storage allocation instance I with maximum job height d, $L_I(t) \leq L' = dn$ for all t, and $\Delta = OPT - LOAD \geq \frac{n^{1/2-1/\lceil d/2 \rceil}}{12\sqrt{d}}$. Because $n = \frac{L-r}{d} \leq \frac{L}{d}$, it follows that $L_I(t) \leq L$ for all t; and because $n \geq \frac{L}{2d}$, it follows that

$$\Delta \geq \frac{n^{1/2-1/\lceil d/2 \rceil}}{12\sqrt{d}} \geq \frac{\left(\frac{L}{2d}\right)^{1/2-1/\lceil d/2 \rceil}}{12\sqrt{d}} = \frac{1}{12\sqrt{d}} \left(\frac{1}{2d}\right)^{1/2-1/\lceil d/2 \rceil} L^{1/2-1/\lceil d/2 \rceil}.$$

The claim follows by setting

$$c = \frac{1}{12\sqrt{d}} \left(\frac{1}{2d}\right)^{1/2-1/\lceil d/2 \rceil}$$

and noting that $1/\lceil d/2 \rceil \leq \epsilon$. □

COROLLARY 6.5. *For any $\epsilon > 0$, there exists a $c' > 0$ such that for all sufficiently large integers $h_{\max}$, there is a* dynamic storage allocation *instance with maximum job*

height $h_{\max}$, maximum load at most L, and $OPT - LOAD \geq c'(h_{\max}/L)^{1/2+\epsilon}L$, for infinitely many integers L.

Proof. Given $\epsilon > 0$, define $h = h(\epsilon)$ and $c > 0$ so that for all sufficiently large L_0, Corollary 6.4 yields a dynamic storage allocation instance with maximum job height h, maximum load at most L_0, and optimum makespan $OPT_0 \geq L_0 + cL_0^{1/2-\epsilon}$. Set $c' = c/h^{1/2+\epsilon}$. For all integers L and $h_{\max}$ such that L and $L/h_{\max}$ are sufficiently large and $h_{\max}/h$ and $L/(h_{\max}/h)$ are integral, set $L_0 = L/(h_{\max}/h)$. Now scale up the instance by $h_{\max}/h$. The new optimum makespan is $OPT = (h_{\max}/h)OPT_0$. The new load is at most $L_0 h_{\max}/h = L$, and

$$\begin{aligned} OPT &= (h_{\max}/h)OPT_0 \\ &\geq (h_{\max}/h)(L_0 + cL_0^{1/2-\epsilon}) \\ &= L + (h_{\max}/h)cL_0^{1/2-\epsilon} \\ &= L(1 + (h_{\max}/(hL))cL_0^{1/2-\epsilon}) \\ &= L(1 + (h_{\max}/(hL))c(hL/h_{\max})^{1/2-\epsilon}) \\ &= L(1 + (c/h^{1/2+\epsilon})(h_{\max}/L)^{1/2+\epsilon}) \\ &= L(1 + c'(h_{\max}/L)^{1/2+\epsilon}). \qquad \square \end{aligned}$$

Acknowledgments. We thank the anonymous referees for several helpful comments.

REFERENCES

[1] M. Chrobak and M. Slusarek, *On some packing problem related to dynamic storage allocation*, RAIRO Inform. Theor. Appl., 22 (1988), pp. 487–499.

[2] M. de Berg, M. van Krevald, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, Berlin, 2000.

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

[4] J. Gergov, *Approximation algorithms for dynamic storage allocation*, in Algorithms—ESA 1996, Lecture Notes in Comput. Sci. 1136, J. Diaz and M. J. Serna, eds., Springer-Verlag, Berlin, 1996, pp. 52–61.

[5] J. Gergov, *Algorithms for compile-time memory optimization*, in Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, SIAM, Philadelphia, 1999, pp. 907–908.

[6] H. A. Kierstead, *The linearity of first-fit coloring of interval graphs*, SIAM J. Discrete Math., 1 (1988), pp. 526–530.

[7] H. A. Kierstead, *A polynomial time approximation algorithm for dynamic storage allocation*, Discrete Math., 88 (1991), pp. 231–237.

[8] D. R. Woodall, *Problem no.* 4, in Proceedings of the British Combinatorial Conference (1973), London Math. Soc. Lecture Note. Ser. 13, Cambridge University Press, Cambridge, UK, 1974, p. 202.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 647–658

DISTANCES AND FINGER SEARCH IN RANDOM BINARY SEARCH TREES*

LUC DEVROYE† AND RALPH NEININGER‡

Abstract. For the random binary search tree with n nodes inserted the number of ancestors of the elements with ranks k and ℓ, $1 \le k < \ell \le n$, as well as the path distance between these elements in the tree are considered. For both quantities, central limit theorems for appropriately rescaled versions are derived. For the path distance, the condition $\ell - k \to \infty$ as $n \to \infty$ is required. We obtain tail bounds and the order of higher moments for the path distance. The path distance measures the complexity of finger search in the tree.

Key words. random binary search tree, finger search, path distance, limit law, analysis of algorithms

AMS subject classifications. 60D05, 68U05

DOI. 10.1137/S0097539703424521

1. Introduction and results. In this paper we analyze the asymptotic behavior of the path distance between nodes in random binary search trees. The path distance between two nodes is the number of nodes on the shortest path connecting them in the tree. This quantity is motivated by the cost of a finger search in the tree. The finger search operation in a search tree takes as input a pointer to a node u, the current node, and either the key value of another node v or an incremental rank value Δ. The objective is to find v quickly. In the latter case, the rank of v differs from the rank of u by Δ. Finger search trees are search trees in which the finger operation takes time $O(1 + \ln \Delta)$. Various strategies are known for this. For example, Brown and Tarjan [2] recommend (2, 4) or red-black trees with level linking. Huddleston and Mehlhorn [6] show how to update these trees efficiently in an amortized sense. On pointer-based machines, Brodal [1] shows how to implement insertion in constant worst-case time in an adaptation of these trees.

In a random binary search tree or a treap, suitably augmented but without level linking, we note that both kinds of finger search operations take time proportional to the path distance between the nodes. The augmentation consists of maintaining with each node either the minimum and maximum keys in the subtree, or the size of the subtree. These parameters are easy to update. Furthermore, when searching for v, starting from u, one first proceeds by following parent pointers towards the root until the least common ancestor of u and v is found. At that point, one can find v by the standard search operation.

If the nodes are level-linked, then it is also possible to identify an ancestor of v that is either the least common ancestor of u and v, or a descendant of that least common ancestor, simply by checking the key values of the appropriate level neighbors of the ancestors of u when traveling towards the root. In this implementation, the

*Received by the editors March 20, 2003; accepted for publication (in revised form) November 17, 2003; published electronically March 30, 2004.

http://www.siam.org/journals/sicomp/33-3/42452.html

†School of Computer Science, McGill University, 3480 University Street, Montreal H3A 2K6, Canada (luc@cs.mcgill.ca). The research of this author was supported by NSERC grant A3450.

‡Department of Mathematics, J.W. Goethe University, Robert-Mayer-Str. 10, 60325 Frankfurt a.M., Germany (neiningr@math.uni-frankfurt.de). The research of this author was supported by DFG grant Ne 828/1-2 (Emmy Noether Programme).

complexity of the finger search operation is the path distance between u and v or less. Other possible augmentations for treaps are presented by Seidel and Aragon [14].

We give an approach of the distributional analysis of the path distance between nodes in a random binary search tree whose keys have ranks that differ by Δ. The connection used between records and random permutations for the study of random binary search trees was developed in Devroye [4] and, when it applies, leads to short and intuitive proofs. While the expectation of the path distance of two nodes that hold keys with ranks differing by Δ is always $O(\ln \Delta)$ as $\Delta \to \infty$, for a refined distributional analysis the location of the ranks matters, since in particular the leading constant in the expansion of the expectation of the path distance depends upon the location of the ranks. This affects the proper scaling of the quantities to obtain distributional convergence; see Theorem 1.3 below.

For simplicity we assume that the random binary search tree is built up from the keys $1, \dots, n$ identifying the key of rank j with the key j. See, e.g., Mahmoud [8] for the definition of random binary search trees. For $1 \le k \le \ell \le n$ we denote by $A_{k\ell}$ the number of ancestors of the nodes holding the keys k and ℓ in the tree when n numbers are inserted. Note that A_{kk} is the depth of the node with rank k in the tree, $1 \le k \le n$. By $P_{k\ell}$ the path distance between the keys k and ℓ is denoted, that is, the number of nodes on the path (strictly) between k and ℓ, $1 \le k < \ell \le n$.

We denote by $\mathcal{N}(0,1)$ the standard normal distribution and by $\xrightarrow{\mathcal{L}}$ convergence in distribution. For sequences $(a_n), (b_n)$ asymptotical equivalence, $a_n/b_n \to 1$ as $n \to \infty$, is denoted by $a_n \sim b_n$. We have the following asymptotic behavior.

THEOREM 1.1. *For all $1 \le k < \ell \le n$, where k, ℓ may depend on n, we have, as $n \to \infty$,*

$$\begin{aligned} \mathbb{E}\, A_{k\ell} &= \ln(k(\ell-k)^2(n-\ell+1)) + O(1), \\ \frac{A_{k\ell} - \ln(k(\ell-k)^2(n-\ell+1))}{\sqrt{\ln(k(\ell-k)^2(n-\ell+1))}} &\xrightarrow{\mathcal{L}} \mathcal{N}(0,1). \end{aligned}$$

THEOREM 1.2. *For all $1 \le k \le n$, where k may depend on n, we have, as $n \to \infty$,*

$$\frac{A_{kk} - \ln(k(n-k+1))}{\sqrt{\ln(k(n-k+1))}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

THEOREM 1.3. *For all $1 \le k < \ell \le n$ with k, ℓ depending on n such that $\Delta := \ell - k + 1 \to \infty$ as $n \to \infty$ and $a_n := (k \wedge \Delta)\Delta^2((n-\ell+1) \wedge \Delta)$ we have, as $n \to \infty$,*

$$\frac{P_{k\ell} - \ln a_n}{\sqrt{\ln a_n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

THEOREM 1.4. *Let P_n denote the path distance between a pair of nodes chosen uniformly at random from all possible pairs of different nodes in the tree. Then we have, as $n \to \infty$,*

$$\frac{P_n - 4\ln n}{\sqrt{4\ln n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

THEOREM 1.5. *There exists a constant $C > 0$ such that for all $\varepsilon > 0$ and all $1 \le k < \ell \le n$ with $\Delta := \ell - k + 1 \ge \Delta_0$ we have with $a_n := (k \wedge \Delta)\Delta^2((n-\ell+1) \wedge \Delta)$:*

$$\mathbb{P}(P_{k\ell} > (1+\varepsilon)\ln a_n) \le C\Delta^{-\varepsilon^2/(2+3\varepsilon)}.$$

Here, for all $\delta > 0$, we can choose $\Delta_0 \ge 1$ uniformly for all $\varepsilon \in [\delta, \infty)$.

Moreover, if $\Delta \to \infty$ *as* $n \to \infty$, *we have, for all* $p \geq 1$,

$$\mathbb{E}\, P_{k\ell}^p \sim \ln^p a_n.$$

Note that exact expressions for $\mathbb{E}\, A_{kk}$ and $\mathbb{E}\, P_{k\ell}$ in terms of harmonic numbers are given in Seidel and Aragon [14] and, for $\mathbb{E}\, A_{k\ell}$, in Prodinger [13]. The limit law in Theorem 1.4, together with additional results for the model of uniformly chosen pairs of nodes, has been derived in Mahmoud and Neininger [9] and Panholzer and Prodinger [12], and an exact expression for $\mathbb{E}\, P_n$ has first been given in Flajolet, Ottmann, and Wood [5]. Finally, we note that the limit law for the depth of a typical node inserted in a random binary search tree was obtained by Mahmoud and Pittel [10], Louchard [7], and Devroye [4]. It can be obtained from Theorem 1.2 by replacing k with a uniform$\{1, \dots, n\}$ random variable.

2. Representation via records. In a permutation $(x_1, \dots, x_n)$ of distinct numbers we define the local ranks $R_1, \dots, R_n$, where R_j denotes the rank of x_j in $\{x_1, \dots, x_j\}$. If $R_j = j$ or $R_j = 1$, we say that x_j is an up-record or down-record in $x_1, \dots, x_n$, respectively. It is well known that if the permutation is a random permutation, i.e., all $n!$ permutations are equally likely, R_j is uniformly distributed on $\{1, \dots, j\}$ for all $j = 1, \dots, n$ and that $R_1, \dots, R_n$ are independent.

We give a representation of the number $A_{k\ell}$ of ancestors of keys k and ℓ in terms of local ranks and records, so that based on the independence properties we can apply the classical central limit theorem in the version of Lindeberg–Feller.

Let us build up the random binary search tree from the numbers $1, \dots, n$ as follows: We draw independent unif$[0,1]$ random variables $T_1, \dots, T_n$, where unif$[0,1]$ denotes the uniform distribution on the interval $[0,1]$. These we use as time stamps as T_j is associated with j and denotes the time at which number j is inserted into the tree. Inserting now the numbers in order according to their time stamps, starting with the earliest, yields a random binary search tree for the keys $1, \dots, n$.

A basic property of the binary search tree is that j is an ancestor of k in the tree if and only if it is inserted before k and also before all numbers s between j and k. Now we fix $1 \leq k < \ell \leq n$ and count the ancestors $A_{k\ell}$ of the elements k and ℓ in the tree. If, for $i < k$, element i is ancestor of ℓ, then it is ancestor of k as well and hence it contributes to $A_{k\ell}$ if and only if

$$T_i = \min\{T_i, T_{i+1}, \dots, T_k\}, \quad i < k.$$

Analogously, for $i > \ell$, we get a contribution of number i to $A_{k\ell}$ if and only if $T_i = \min\{T_\ell, T_{\ell+1}, \dots, T_i\}$, and in the case $k < i < \ell$ if $T_i = \min\{T_k, T_{k+1}, \dots, T_i\}$ or $T_i = \min\{T_i, T_{i+1}, \dots, T_\ell\}$. Passing to indicator functions we rewrite these events as

$$\begin{aligned}
\mathbf{1}_{\{T_i = \min\{T_i, T_{i+1}, \dots, T_k\}\}} &= \mathbf{1}_{\{T_i = \min\{T_i, \dots, T_{k-1}\}\}} - \mathbf{1}_{\{T_k < T_i,\, T_i = \min\{T_i, \dots, T_{k-1}\}\}} \\
&=: \mathbf{1}_{B_i} - \mathbf{1}_{C_i}, \quad i < k, \\
\mathbf{1}_{\{T_i = \min\{T_\ell, \dots, T_i\}\}} &= \mathbf{1}_{\{T_i = \min\{T_{\ell+1}, \dots, T_i\}\}} - \mathbf{1}_{\{T_\ell < T_i,\, T_i = \min\{T_{\ell+1}, \dots, T_i\}\}} \\
&=: \mathbf{1}_{B_i} - \mathbf{1}_{C_i}, \quad i > \ell,
\end{aligned}$$

and

$$\mathbf{1}_{B_i} := \mathbf{1}_{\{T_i = \min\{T_k, T_{k+1}, \dots, T_i\}\} \cup \{T_i = \min\{T_i, T_{i+1}, \dots, T_\ell\}\}}, \quad k \leq i \leq \ell.$$

Note that above $\mathbf{1}_{B_i}, \mathbf{1}_{C_i}$ are differently defined for the three ranges of the index i.

All together we obtain the representation

$$A_{k\ell} = \sum_{i=1}^{n} \mathbf{1}_{B_i} - \sum_{i=1}^{k-1} \mathbf{1}_{C_i} - \sum_{i=\ell+1}^{n} \mathbf{1}_{C_i} - 2, \tag{2.1}$$

where we subtract 2 referring to the convention that k and ℓ are not counted as ancestors of themselves. The main contribution comes from the sum over the $\mathbf{1}_{B_i}$, as the sums over the $\mathbf{1}_{C_i}$ will be asymptotically negligible.

To get the connection with records we introduce three auxiliary random binary search trees as follows. The binary search tree $\mathcal{T}_<$ is built up from the elements $1, \dots, k-1$, inserted according to their time stamps $T_1, \dots, T_{k-1}$. Analogously $\mathcal{T}_>$ is built up from the elements $\ell+1, \dots, n$, inserted according to their time stamps $T_\ell, \dots, T_n$ and $\mathcal{T}$ is built up from the elements $k, \dots, \ell$, inserted according to their time stamps $T_k, \dots, T_\ell$. Now, for $i < k$, the event B_i is equivalent for i to be an ancestor of $k-1$ in $\mathcal{T}_<$. Since $k-1$ is the largest element in $\mathcal{T}_<$, this implies that i is an up-record at the time of insertion into $\mathcal{T}_<$. Analogously, for $i > \ell$, the event B_i is equivalent for i to constitute a down-record at time of its insertion into $\mathcal{T}_>$. For $k \le i \le \ell$, event B_i is equivalent to i being up- or down-record at its time of insertion into $\mathcal{T}$.

We denote by R_j the local rank of the (in time) jth element inserted into $\mathcal{T}_<$ at the time of its insertion, $1 \le j < k$, and by R'_j, R''_j analogously the local ranks of the jth elements inserted into $\mathcal{T}$, $\mathcal{T}_>$ for $1 \le j \le \ell-k+1$ and $1 \le j \le n-\ell$, respectively. Note that $R_1, \dots, R_{k-1}, R'_1, \dots, R'_{\ell-k+1}, R''_1, \dots, R''_{n-\ell}$ are independent and R_j, R'_j, R''_j are uniform$\{1, \dots, j\}$ distributed for $j = 1, \dots, k-1$ and $j = 1, \dots, \ell-k+1$ and $j = 1, \dots, n-\ell$, respectively. We have

$$\sum_{i=1}^{n} \mathbf{1}_{B_i} = \sum_{j=1}^{k-1} \mathbf{1}_{\{R_j = j\}} + \sum_{j=1}^{\ell-k+1} \mathbf{1}_{\{R'_j \in \{1,j\}\}} + \sum_{j=1}^{n-\ell} \mathbf{1}_{\{R''_j = 1\}}. \tag{2.2}$$

For the representation of $P_{k\ell}$ we denote

$$T_A := \min\{T_k, \dots, T_\ell\}.$$

For $1 \le i \le n$, element i belongs to the path between k and ℓ if and only if it is ancestor of k or ℓ and $T_i \ge T_A$. Hence with $D_i := \{T_i \ge T_A\}$ we have

$$P_{k\ell} = \sum_{i=1}^{n} \mathbf{1}_{B_i \cap D_i} - \sum_{i=1}^{k-1} \mathbf{1}_{C_i \cap D_i} - \sum_{i=\ell+1}^{n} \mathbf{1}_{C_i \cap D_i} - 2. \tag{2.3}$$

The main contribution will come from the sum over the $\mathbf{1}_{B_i \cap D_i}$. For the corresponding representation with records we introduce

$$N_1 := |\{1 \le j < k : T_j < T_A\}|, \quad N_2 := |\{\ell < j \le n : T_j < T_A\}|$$

and obtain

$$\begin{aligned} \sum_{i=1}^{n} \mathbf{1}_{B_i \cap D_i} &= \sum_{j=N_1+1}^{k-1} \mathbf{1}_{\{R_j = j\}} + \sum_{j=1}^{\ell-k+1} \mathbf{1}_{\{R'_j \in \{1,j\}\}} + \sum_{j=N_2+1}^{n-\ell} \mathbf{1}_{\{R''_j = 1\}} \\ &=: \mathcal{P}_I + \mathcal{P}_{II} + \mathcal{P}_{III}. \end{aligned} \tag{2.4}$$

3. Proofs. Throughout this section we denote by $H_n := \sum_{i=1}^{n} 1/i = \ln n + O(1)$ the nth harmonic number for $n \geq 1$ and $H_0 := 0$.

Proof of Theorem 1.1. We derive $\mathbb{E}\, A_{k\ell}$ using the representations (2.1) and (2.2). From the distribution of the local ranks R_j, R'_j, and R''_j we obtain

$$\mathbb{E} \sum_{i=1}^{n} \mathbf{1}_{B_i} = H_{k-1} + 2H_{\ell-k+1} - 1 + H_{n-\ell} = \ln(k(\ell-k)^2(n-\ell+1)) + O(1).$$

The remaining summands in (2.1) we denote by $\Upsilon := \sum_{i=1}^{k-1} \mathbf{1}_{C_i} + \sum_{i=\ell+1}^{n} \mathbf{1}_{C_i} + 2$. For $1 \leq i < k$ we have

$$\begin{aligned}
\mathbb{E}\, \mathbf{1}_{C_i} &= \mathbb{P}\Big(T_k < T_i,\ T_i = \min\{T_i, \ldots, T_{k-1}\}\Big) \\
&\leq \mathbb{P}\Big(T_k, T_i \text{ are the smallest elements among } T_i, \ldots, T_k\Big) \\
&= \binom{k-i+1}{2}^{-1} \leq \frac{2}{(k-i)^2}.
\end{aligned}$$

This implies $\mathbb{E} \sum_{i=1}^{k-1} \mathbf{1}_{C_i} = O(1)$. Analogously we conclude to find $\mathbb{E}\, \Upsilon = O(1)$, hence we obtain $\mathbb{E}\, A_{k\ell} = \ln(k(\ell-k)^2(n-\ell+1)) + O(1)$.

For the central limit law we write

$$\begin{aligned}
\frac{A_{k\ell} - \ln(k(\ell-k)^2(n-\ell+1))}{\sqrt{\ln(k(\ell-k)^2(n-\ell+1))}} &= \frac{\sum_{i=1}^{n} \mathbf{1}_{B_i} - \ln(k(\ell-k)^2(n-\ell+1))}{\sqrt{\ln(k(\ell-k)^2(n-\ell+1))}} \\
&\quad - \frac{\Upsilon}{\sqrt{\ln(k(\ell-k)^2(n-\ell+1))}}.
\end{aligned}$$

For all choices of $1 \leq k < \ell \leq n$ we have $\ln(k(\ell-k)^2(n-\ell+1)) \to \infty$ as $n \to \infty$, and from (2.2) it follows that the Lindeberg–Feller condition (see Chow and Teicher [3, p. 291]) is satisfied for $\sum_{i=1}^{n} \mathbf{1}_{B_i}$; thus the first fraction on the right-hand side of the latter display tends in distribution to the standard normal distribution. Again since $\ln(k(\ell-k)^2(n-\ell+1)) \to \infty$ and $\mathbb{E}\,|\Upsilon| = O(1)$ we obtain from Markov's inequality that $\Upsilon / \ln(k(\ell-k)^2(n-\ell+1)) \to 0$ in probability as $n \to \infty$. The assertion follows. □

Proof of Theorem 1.2. Note that for A_{kk} we have the same representation as for $A_{k\ell}$ given, for the case $k < \ell$, in (2.1), where we have to replace the -2 there by -1 due to the fact that we now have $\mathbf{1}_{B_k} = 1$. Hence the same arguments as in the proof of Theorem 1.1 apply. □

Proof of Theorem 1.3. We have $P_{k\ell} = \mathcal{P}_I + \mathcal{P}_{II} + \mathcal{P}_{III} - \Upsilon'$, with $\Upsilon' := \sum_{i=1}^{k-1} \mathbf{1}_{C_i \cap D_i} + \sum_{i=\ell+1}^{n} \mathbf{1}_{C_i \cap D_i} + 2$ and $a_n := (k \wedge \Delta)\Delta^2((n-\ell+1) \wedge \Delta) \to \infty$ as $n \to \infty$. From $\mathbb{E}\,|\Upsilon'| = O(1)$ we obtain from Markov's inequality $\Upsilon' / \sqrt{\ln a_n} \to 0$ in probability. Thus it is sufficient to show

$$\frac{\mathcal{P}_I + \mathcal{P}_{II} + \mathcal{P}_{III} - \ln a_n}{\sqrt{\ln a_n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1). \tag{3.1}$$

Since we want to apply the central limit theorem to the sum of indicators in (2.4) we will condition on the random indices N_1 and N_2. Note that we may assume $k \to \infty$ and $n - \ell + 1 \to \infty$ as $n \to \infty$ since otherwise $\mathcal{P}_I$ and $\mathcal{P}_{III}$ remain bounded and do not contribute respectively.

First we consider the case $k/\Delta > \ln k$ and $(n-\ell+1)/\Delta > \ln(n-\ell+1)$ for all sufficiently large n. We define, for $\varepsilon > 0$,

$$B_\varepsilon := \{N_1 \in [\alpha_1, \beta_1]\} \cap \{N_2 \in [\alpha_2, \beta_2]\},$$

with

$$\alpha_1 = \frac{\varepsilon}{2}\frac{k}{\Delta}, \quad \beta_1 = \frac{2}{\varepsilon}\frac{k}{\Delta}, \qquad \alpha_2 = \frac{\varepsilon}{2}\frac{n-\ell+1}{\Delta}, \quad \beta_2 = \frac{2}{\varepsilon}\frac{n-\ell+1}{\Delta}.$$

Note that the values of N_1 and N_2 depend on $T_k, \dots, T_\ell$. However, conditioned on N_1 and N_2 the permutations induced by $T_1, \dots, T_{k-1}$, by $T_k, \dots, T_\ell$, and by $T_{\ell+1}, \dots, T_n$ are independent and uniformly distributed. In particular, conditioning on N_1, N_2 preserves the independence and the distributions of $R_1, \dots, R_{k-1}, R'_1, \dots, R'_\Delta, R''_1, \dots, R_{n-\ell}$.

On B_ε we have the bounds $P^-_{k\ell} \le P_{k\ell} \le P^+_{k\ell}$ with

$$P^-_{k\ell} = \sum_{j=\lceil\beta_1\rceil+1}^{k-1} \mathbf{1}_{\{R_j=j\}} + \sum_{j=1}^{\Delta} \mathbf{1}_{\{R'_j\in\{1,j\}\}} + \sum_{j=\lceil\beta_2\rceil+1}^{n-\ell} \mathbf{1}_{\{R''_j=1\}},$$
$$P^+_{k\ell} = \sum_{j=\lfloor\alpha_1\rfloor}^{k-1} \mathbf{1}_{\{R_j=j\}} + \sum_{j=1}^{\Delta} \mathbf{1}_{\{R'_j\in\{1,j\}\}} + \sum_{j=\lfloor\alpha_2\rfloor}^{n-\ell} \mathbf{1}_{\{R''_j=1\}}.$$

Now, we have

$$\begin{aligned}\mathbb{E}\, P^-_{k\ell} &= \ln k - \ln\lceil\beta_1\rceil + 2\ln\Delta \\ &\quad + \ln(n-\ell+1) - \ln\lceil\beta_2\rceil + O(1) \\ &= \ln a_n + O\Big(1 + \ln\frac{1}{\varepsilon}\Big),\end{aligned} \tag{3.2}$$

where, for the last equality, we distinguish the cases $k/\Delta \le 2/\varepsilon$, $k/\Delta > 2/\varepsilon$ as well as $(n-\ell+1)/\Delta \le 2/\varepsilon$ and $(n-\ell+1)/\Delta > 2/\varepsilon$. Analogously we obtain $\operatorname{Var}(P^-_{k\ell}) = \mathbb{E}\, P^-_{k\ell} + O(1+\ln(1/\varepsilon))$. Since $\varepsilon > 0$ is fixed and $a_n \to \infty$ as $n \to \infty$ we obtain from the central limit theorem in the version of Lindeberg–Feller that

$$\frac{P^-_{k\ell} - \ln a_n}{\sqrt{\ln a_n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1), \quad n \to \infty. \tag{3.3}$$

Similarly, we obtain $(P^+_{k\ell} - \ln a_n)/\sqrt{\ln a_n} \to \mathcal{N}(0,1)$ in distribution as $n \to \infty$. We have, for $x \in \mathbb{R}$,

$$\begin{aligned}\mathbb{P}\Big(\frac{P_{k\ell} - \ln a_n}{\sqrt{\ln a_n}} \le x\Big) &\le \mathbb{P}(B^c_\varepsilon) + \mathbb{P}\Big(\frac{P_{k\ell} - \ln a_n}{\sqrt{\ln a_n}} \le x \Big| B_\varepsilon\Big) \\ &\le \mathbb{P}(B^c_\varepsilon) + \mathbb{P}\Big(\frac{P^-_{k\ell} - \ln a_n}{\sqrt{\ln a_n}} \le x\Big).\end{aligned}$$

Hence denoting by Φ the distribution function of the standard normal distribution and $\psi(\varepsilon) := \limsup_{n\to\infty} \mathbb{P}(B^c_\varepsilon)$ we obtain

$$\limsup_{n\to\infty} \mathbb{P}\Big(\frac{P_{k\ell} - \ln a_n}{\sqrt{\ln a_n}} \le x\Big) \le \Phi(x) + \psi(\varepsilon),$$

and analogously

$$\begin{aligned}\liminf_{n\to\infty} \mathbb{P}\Big(\frac{P_{k\ell} - \ln a_n}{\sqrt{\ln a_n}} \le x\Big) &\ge \liminf_{n\to\infty} \mathbb{P}(B_\varepsilon)\mathbb{P}\Big(\frac{P^+_{k\ell} - \ln a_n}{\sqrt{\ln a_n}} \le x\Big) \\ &= (1-\psi(\varepsilon))\Phi(x).\end{aligned}$$

Hence the central limit law is established once we have shown that $\psi(\varepsilon) \to 0$ as $\varepsilon \downarrow 0$. For this it is sufficient to show that $[\limsup_{n\to\infty} \mathbb{P}(N_i \notin [\alpha_i, \beta_i])] \to 0$ as $\varepsilon \downarrow 0$ for $i = 1, 2$. By symmetry we only need to show the case $i = 1$.

We denote by $B_{n,u}$ a binomial $B(n, u)$ distributed random variable, $n \geq 0$, $u \in [0, 1]$. Since N_1 has the mixed $B(k-1, T_A)$ distribution with $T_A = \min\{T_k, \ldots, T_\ell\}$ we obtain with Chebyshev's inequality, for $k \geq 4$ and Δ sufficiently large such that $\varepsilon/\Delta \leq 1$,

$$\begin{aligned}
\mathbb{P}\Big(N_1 < \frac{\varepsilon k}{2\Delta}\Big) &\leq \mathbb{P}\Big(T_A < \frac{\varepsilon}{\Delta}\Big) + \mathbb{P}\Big(B_{k-1,\varepsilon/\Delta} \leq \frac{\varepsilon k}{2\Delta}\Big) \\
&\leq 2\varepsilon + \mathbb{P}\Big(\Big|B_{k-1,\varepsilon/\Delta} - \frac{\varepsilon(k-1)}{\Delta}\Big| \geq \frac{\varepsilon k}{4\Delta}\Big) \\
&\leq 2\varepsilon + \frac{16}{\varepsilon(k/\Delta)} \\
&\leq 2\varepsilon + \frac{16}{\varepsilon \ln k} \\
&\to 2\varepsilon
\end{aligned}$$

as $n \to \infty$. Similarly we obtain for sufficiently large Δ,

$$\begin{aligned}
\mathbb{P}\Big(N_1 > \frac{2k}{\varepsilon\Delta}\Big) &\leq \mathbb{P}\Big(T_A > \frac{1}{\varepsilon\Delta}\Big) + \mathbb{P}\Big(B_{k-1,1/(\varepsilon\Delta)} \geq \frac{2k}{\varepsilon\Delta}\Big) \\
&\leq 2e^{-1/\varepsilon} + \frac{\varepsilon}{\ln k} \\
&\to 2e^{-1/\varepsilon}
\end{aligned}$$

as $n \to \infty$. Hence we obtain $[\limsup_{n\to\infty} \mathbb{P}(N_1 \notin [\alpha_1, \beta_1])] \leq 2(\varepsilon + e^{-1/\varepsilon}) \to 0$ as $\varepsilon \downarrow 0$.

In the second case we assume that $k/\Delta \leq \ln k$ and $(n - \ell + 1)/\Delta > \ln(n - \ell + 1)$ for all n sufficiently large. Now we replace α_i, β_i with

$$\alpha_1' = 0, \quad \beta_1' = \ln^2 k, \qquad \alpha_2' = \alpha_2, \quad \beta_2' = \beta_2$$

and define $B_\varepsilon, P_{k\ell}^-, P_{k\ell}^+$ as in the first case but with the α_i, β_i replaced by α_i', β_i', $i = 1, 2$. The argument is now applied as in the first case. The only difference to be shown is that we have $\limsup_{n\to\infty} \mathbb{P}(N_1 \notin [\alpha_1', \beta_1']) = 0$: We have

$$\mathbb{P}(N_1 \notin [\alpha_1', \beta_1']) = \mathbb{P}(N_1 > \ln^2 k) \leq \frac{\mathbb{E}\, N_1}{\ln^2 k} = \frac{(k-1)/\Delta}{\ln^2 k} \leq \frac{1}{\ln k} \to 0$$

as $n \to \infty$.

The case $k/\Delta > \ln k$ and $(n - \ell + 1)/\Delta \leq \ln(n - \ell + 1)$ is covered by the previous case by symmetry. In the remaining case $k/\Delta \leq \ln k$ and $(n - \ell + 1)/\Delta \leq \ln(n - \ell + 1)$ we replace α_i, β_i with

$$\alpha_1'' = \alpha_1', \quad \beta_1'' = \beta_1', \qquad \alpha_2'' = 0, \quad \beta_2'' = \ln(n - \ell + 1)^2,$$

and define $B_\varepsilon, P_{k\ell}^-, P_{k\ell}^+$ as in the first case but with the α_i, β_i replaced by α_i'', β_i'', $i = 1, 2$. The argument is again applied as in the first case and $\limsup_{n\to\infty} \mathbb{P}(N_i \notin [\alpha_i'', \beta_i'']) = 0$ follows for $i = 1, 2$ as in the second case.

This finishes the proof of the limit law since for a given sequence $(k,\ell)=(k(n),\ell(n))$ with $\ell(n)-k(n)\to\infty$ we decompose into four subsequences according to whether $k/\Delta\le\ln k$ or $k/\Delta>\ln k$ and $(n-\ell+1)/\Delta\le\ln(n-\ell+1)$ or $(n-\ell+1)/\Delta>\ln(n-\ell+1)$. Each of the subsequences satisfies, by the previous arguments, the limit law (3.1), hence the whole sequence satisfies the limit law. □

Proof of Theorem 1.4. We denote by (K,L) the ranks of the pair of nodes chosen uniformly at random from all possible pairs of distinct nodes in the tree, where we may assume that $K<L$. We define the set

$$B:=\Big\{K<\frac{n}{\ln n}\Big\}\cup\Big\{n-L<\frac{n}{\ln n}\Big\}\cup\Big\{L-K<\frac{n}{\ln n}\Big\}$$

and note that $\mathbb{P}(B)\to 0$ as $n\to\infty$. On B^c we will condition on $(K,L)=(k,\ell)$. For these (k,ℓ) we have $\ln(k(\ell-k+1)^2(n-\ell+1))=4\ln n+O(\ln\ln n)$. Hence application of Theorem 1.3 yields $(P_{k\ell}-4\ln n)/\sqrt{4\ln n}\to\mathcal{N}(0,1)$ in distribution. Denoting by Φ the distribution function of $\mathcal{N}(0,1)$ and by σ the distribution of (K,L) we obtain, for all $x\in\mathbb{R}$,

$$\begin{aligned}&\left|\mathbb{P}\left(\frac{P_n-4\ln n}{\sqrt{4\ln n}}\le x\right)-\Phi(x)\right|\\&\le\mathbb{P}(B)+\int\left|\mathbb{P}\left(\frac{P_{k\ell}-4\ln n}{\sqrt{4\ln n}}\le x\right)-\Phi(x)\right|d\sigma(k,\ell)\\&\to 0\end{aligned}$$

by dominated convergence. The assertion follows. □

To prepare for the proof of Theorem 1.5 we provide the following tail estimate:

LEMMA 3.1. *Let Y_j, $1\le j\le n$ be independent and Y_j be Bernoulli $B(p_j)$ distributed for $0\le p_j\le 1$, and $\mu=\sum_{j=1}^n p_j$. Then we have, for all $\varepsilon>0$,*

$$\mathbb{P}\left(\sum_{j=1}^n Y_j\ge\mu+\varepsilon\right)\le\exp\Big(-\frac{\varepsilon^2}{2\mu+\varepsilon}\Big).$$

Proof. The proof relies on Chernoff's bounding technique. The details follow the proof of Theorem L1 in Devroye [4]. □

COROLLARY 3.2. *Let X_j,X_j' be Bernoulli $B(1/j)$ distributed, $j\ge 1$, $Z_1=1$ and Z_j be $B(2/j)$ distributed, $j\ge 2$, such that all random variables are independent. Then for all $1\le q\le s$, $\Delta\ge 1$, $1\le r\le t$ we have with $\alpha:=s\Delta^2t/(qr)$,*

$$\mathbb{P}\left(\sum_{j=q}^s X_j+\sum_{j=1}^{\Delta}Z_j+\sum_{j=r}^t X_j'-\ln\alpha\ge\varepsilon\right)\le\exp\left(-\frac{(\varepsilon-7)^2}{\varepsilon+6+2\ln\alpha}\right).$$

Proof. We apply Lemma 3.1 and note that from $\ln(n+1)\le H_n\le 1+\ln n$ for $n\ge 1$, we obtain

$$\ln(\alpha)-7\le H_s-H_{q-1}+2H_\Delta-1+H_t-H_{r-1}\le\ln(\alpha)+3.$$

The assertion follows. □

Proof of Theorem 1.5. First we prove the tail bound, where we distinguish several cases for the ranges of k and $n-\ell+1$. We abbreviate a_n as in Theorem 1.5. Let ε be arbitrarily given.

For $k \geq \Delta^{1+\varepsilon}$, $n-\ell+1 \geq \Delta^{1+\varepsilon}$ we have with the representations (2.3) and (2.4) and X_j, X'_j, and Z_j as in Corollary 3.2

$$\begin{aligned}
&\mathbb{P}(P_{k\ell} > (1+\varepsilon)\ln a_n)\\
&\leq \mathbb{P}(\mathcal{P}_I + \mathcal{P}_{II} + \mathcal{P}_{III} > (1+\varepsilon)\ln a_n)\\
&\leq \mathbb{P}\left(\left\{N_1 < \frac{k-1}{\Delta^{1+\varepsilon}}\right\} \cup \left\{N_2 < \frac{n-\ell}{\Delta^{1+\varepsilon}}\right\}\right)\\
&\quad + \mathbb{P}\left(\sum_{j=\lfloor (k-1)/\Delta^{1+\varepsilon}\rfloor+1}^{k-1} X_j + \sum_{j=1}^{\Delta} Z_j + \sum_{j=\lfloor (n-\ell)/\Delta^{1+\varepsilon}\rfloor+1}^{n-\ell} X'_j > (1+\varepsilon)\ln a_n\right).
\end{aligned} \tag{3.4}$$

Using that N_1 is $B(k-1, T_A)$ distributed and $T_A = \min\{T_k, \ldots, T_\ell\}$, we obtain

$$\mathbb{P}\left(N_1 < \frac{k-1}{\Delta^{1+\varepsilon}}\right) \leq \mathbb{P}\left(T_A < \Delta^{-(1+\varepsilon/2)}\right) + \mathbb{P}\left(B_{k-1,1/\Delta^{1+\varepsilon/2}} < \frac{k-1}{\Delta^{1+\varepsilon}}\right). \tag{3.5}$$

The first summand in (3.5) is bounded by

$$\mathbb{P}\left(T_A < \Delta^{-(1+\varepsilon/2)}\right) = 1 - (1-\Delta^{-(1+\varepsilon/2)})^{\Delta} \leq \Delta^{-\varepsilon/2}.$$

For the second summand in (3.5) we use Okamoto's inequality [11], which states that $\mathbb{P}(B_{n,u} \leq ny) \leq \exp(-n(u-y)^2/(2u(1-u)))$ for all $y \leq u \leq 1/2$. For $y := \Delta^{-(1+\varepsilon)}$ and $u := \Delta^{-(1+\varepsilon/2)}$ we obtain, for Δ sufficiently large,

$$\begin{aligned}
\mathbb{P}\left(B_{k-1,1/\Delta^{1+\varepsilon/2}} < \frac{k-1}{\Delta^{1+\varepsilon}}\right) &\leq \exp\left(-(k-1)\frac{\left(\Delta^{-(1+\varepsilon/2)} - \Delta^{-(1+\varepsilon)}\right)^2}{2\Delta^{-(1+\varepsilon/2)}}\right)\\
&\leq \exp\left(-\frac{k-1}{8\Delta^{1+\varepsilon/2}}\right)\\
&\leq \exp\left(-\frac{k+1}{\Delta^{1+\varepsilon}}\frac{\Delta^{\varepsilon/2}}{24}\right)\\
&\leq \exp\left(-\frac{\Delta^{\varepsilon/2}}{24}\right)\\
&\leq 24\Delta^{-\varepsilon/2},
\end{aligned}$$

where we used that $(k+1)/\Delta^{1+\varepsilon} \geq 1$. Note that for this estimate Δ can be chosen uniformly large for all $\varepsilon \in [\delta, \infty)$, $\delta > 0$. By symmetry we obtain the same bound for $\mathbb{P}(N_2 < (n-\ell)/\Delta^{1+\varepsilon})$. The second summand in (3.4) we estimate with Corollary 3.2

for Δ sufficiently large:

$$\begin{aligned}
&\mathbb{P}\left(\sum_{j=\lfloor (k-1)/\Delta^{1+\varepsilon}\rfloor+1}^{k-1} X_j + \sum_{j=1}^{\Delta} Z_j + \sum_{j=\lfloor (n-\ell)/\Delta^{1+\varepsilon}\rfloor+1}^{n-\ell} X'_j > (1+\varepsilon)\ln a_n\right)\\
&\le \mathbb{P}\left(\sum_{j=\lfloor (k-1)/\Delta^{1+\varepsilon}\rfloor+1}^{k-1} X_j + \sum_{j=1}^{\Delta} Z_j + \sum_{j=\lfloor (n-\ell)/\Delta^{1+\varepsilon}\rfloor+1}^{n-\ell} X'_j - \ln\Delta^{4+2\varepsilon} > 2\varepsilon\ln\Delta\right)\\
&\le \exp\left(-\frac{(2\varepsilon\ln\Delta-7)^2}{2\ln\Delta^{4+2\varepsilon}+6+2\varepsilon\ln\Delta}\right)\\
&\le \exp\left(\frac{28\varepsilon}{8+6\varepsilon} - \frac{4\varepsilon^2}{9+6\varepsilon}\ln\Delta\right)\\
&\le e^5\Delta^{-\varepsilon^2/(3+2\varepsilon)}. \qquad (3.6)
\end{aligned}$$

Collecting the estimates, we obtain $\mathbb{P}(P_{k\ell} \ge (1+\varepsilon)\ln a_n) \le 200\Delta^{-\varepsilon^2/(3+2\varepsilon)}$.

For the case $\Delta \le k \le \Delta^{1+\varepsilon}$ and $n-\ell+1 \ge \Delta^{1+\varepsilon}$ we estimate

$$\begin{aligned}
&\mathbb{P}(P_{k\ell} > (1+\varepsilon)\ln a_n)\\
&\le \mathbb{P}\left(N_2 < \frac{n-\ell}{\Delta^{1+\varepsilon}}\right)\\
&\quad + \mathbb{P}\left(\sum_{j=1}^{\lfloor\Delta^{1+\varepsilon}\rfloor} X_j + \sum_{j=1}^{\Delta} Z_j + \sum_{j=\lfloor (n-\ell)/\Delta^{1+\varepsilon}\rfloor+1}^{n-\ell} X'_j > (1+\varepsilon)\ln a_n\right),
\end{aligned}$$

and both summands can be estimated as in the previous case.

The same estimates apply to the cases $k \ge \Delta^{1+\varepsilon}$ and $\Delta \le n-\ell+1 \le \Delta^{1+\varepsilon}$ as well as $\Delta \le k \le \Delta^{1+\varepsilon}$ and $\Delta \le n-\ell+1 \le \Delta^{1+\varepsilon}$. The remaining cases are where either $k < \Delta$ or $n-\ell+1 < \Delta$. If $k < \Delta$ and $n-\ell+1 \ge \Delta^{1+\varepsilon}$, then

$$\begin{aligned}
&\mathbb{P}(P_{k\ell} > (1+\varepsilon)\ln a_n)\\
&\le \mathbb{P}\left(N_2 < \frac{n-\ell}{\Delta^{1+\varepsilon}}\right)\\
&\quad + \mathbb{P}\left(\sum_{j=1}^{k-1} X_j + \sum_{j=1}^{\Delta} Z_j + \sum_{j=\lfloor (n-\ell)/\Delta^{1+\varepsilon}\rfloor+1}^{n-\ell} X'_j > (1+\varepsilon)\ln a_n\right),
\end{aligned}$$

where the first summand is bounded as before and the second one has the upper bound

$$\begin{aligned}
&\mathbb{P}\left(\sum_{j=1}^{k-1} X_j + \sum_{j=1}^{\Delta} Z_j + \sum_{j=\lfloor (n-\ell)/\Delta^{1+\varepsilon}\rfloor+1}^{n-\ell} X'_j - \ln(k\Delta^{3+\varepsilon}) > 2\varepsilon\ln\Delta\right)\\
&\le \exp\left(-\frac{(2\varepsilon\ln\Delta-7)^2}{2\ln(k\Delta^{3+\varepsilon})+6+2\varepsilon\ln\Delta}\right),
\end{aligned}$$

which leads to the bound given in (3.6) since $k\Delta^{3+\varepsilon} \le \Delta^{4+2\varepsilon}$. For the case $k \le \Delta$

and $\Delta \le n-\ell+1 \le \Delta^{1+\varepsilon}$ we estimate

$$\mathbb{P}(P_{k\ell} > (1+\varepsilon)\ln a_n) \le \mathbb{P}\left(\sum_{j=1}^{k-1} X_j + \sum_{j=1}^{\Delta} Z_j + \sum_{j=1}^{\lfloor \Delta^{1+\varepsilon} \rfloor} X'_j > (1+\varepsilon)\ln a_n\right),$$

and, for the case $k \le \Delta$ and $n-\ell+1 \le \Delta$,

$$\mathbb{P}(P_{k\ell} > (1+\varepsilon)\ln a_n) \le \mathbb{P}\left(\sum_{j=1}^{k-1} X_j + \sum_{j=1}^{\Delta} Z_j + \sum_{j=1}^{n-\ell} X'_j > (1+\varepsilon)\ln a_n\right),$$

and estimate as before. The remaining cases with $n-\ell+1 \le \Delta$ are covered by symmetry.

To show the second claim of the theorem, $\mathbb{E}\,P_{k\ell}^p \sim \ln^p a_n$, we fix $p \ge 1$ and $\delta \in (0,1)$. Then, by the first part, there is a $C > 0$ with $\mathbb{P}(P_{k\ell} \ge (1+\varepsilon)\ln a_n) \le C\Delta^{-\varepsilon^2/(3+2\varepsilon)}$ for all Δ sufficiently large and all $\varepsilon \ge \delta$. We obtain

$$\begin{aligned}
\mathbb{E}\,P_{k\ell}^p &= \mathbb{E}\left[P_{k\ell}^p\left(\mathbf{1}_{\{P_{k\ell} \le (1+\delta)\ln a_n\}} + \mathbf{1}_{\{P_{k\ell} > (1+\delta)\ln a_n\}}\right)\right] \\
&\le (1+\delta)^p \ln^p a_n + \int_{(1+\delta)^p \ln^p a_n}^{\infty} \mathbb{P}(P_{k\ell}^p \ge t)\,dt \\
&\le (1+\delta)^p \ln^p a_n + C\int_{(1+\delta)^p \ln^p a_n}^{\infty} \exp\left(-\frac{\varepsilon^2}{3+2\varepsilon}\ln\Delta\right) dt,
\end{aligned}$$

with $\varepsilon = \varepsilon(t) = (t^{1/p}/\ln a_n) - 1$.

Note that for any convex function $f : [t_0, \infty) \to \mathbb{R}$, $t_0 \in \mathbb{R}$, differentiable in t_0 with $f'(t_0) > 0$, we have

$$\int_{t_0}^{\infty} \exp(-f(t))\,dt \le \frac{\exp(-f(t_0))}{f'(t_0)}.$$

This follows estimating $f(t) \ge f(t_0) + f'(t_0)(t-t_0)$ for all $t \ge t_0$ and evaluating the resulting integral.

Now, the function $f(t) = (\varepsilon^2/(3+2\varepsilon))\ln\Delta$ with $\varepsilon = \varepsilon(t)$ given above and $t_0 = (1+\delta)^p \ln^p a_n$ has the latter form. Hence an explicit calculation yields

$$\begin{aligned}
\int_{(1+\delta)^p \ln^p a_n}^{\infty} \exp\left(-\frac{\varepsilon^2}{3+2\varepsilon}\ln\Delta\right) dt &\le \frac{\exp(-f(t_0))}{f'(t_0)} \\
&= \frac{p(1+\delta)^{p-1}\ln^p a_n}{(6\delta+2\delta^2)\ln\Delta}\Delta^{-\delta^2/(3+2\delta)} \\
&= O\left(\frac{\ln^{p-1}\Delta}{\Delta^{\delta^2/(3+2\delta)}}\right),
\end{aligned}$$

which gives a vanishing contribution as $\Delta \to \infty$.

Hence we obtain

$$\limsup_{n\to\infty} \frac{\mathbb{E}\,P_{k\ell}^p}{\ln^p a_n} \le (1+\delta)^p$$

for all $\delta > 0$; hence $\limsup_{n\to\infty} \mathbb{E}\,P_{k\ell}^p/\ln^p a_n \le 1$.

For the lower bound we choose $c \in \mathbb{R}$. Then for all n sufficiently large such that $a_n > \exp(c^2)$ we have

$$\begin{aligned}\frac{\mathbb{E}\, P_{k\ell}^p}{\ln^p a_n} &\geq \frac{1}{\ln^p a_n}\, \mathbb{E}\left[\mathbf{1}_{\{(P_{k\ell}-\ln a_n)/\sqrt{\ln a_n}\geq c\}} P_{k\ell}^p\right] \\ &\geq \left(1+\frac{c}{\sqrt{\ln a_n}}\right)^p \mathbb{P}\left(\frac{P_{k\ell}-\ln a_n}{\sqrt{\ln a_n}} \geq c\right) \\ &\to 1-\Phi(c)\end{aligned}$$

as $n \to \infty$, by Theorem 1.3, where Φ denotes the distribution function of the standard normal distribution. With $c \to -\infty$ we obtain $\liminf_{n\to\infty} \mathbb{E}\, P_{k\ell}^p / \ln^p a_n \geq 1$. □

Acknowledgment. We thank the referees for their careful reading.

REFERENCES

[1] G. S. Brodal, *Finger search trees with constant insertion time*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 1998), ACM, New York, 1998, pp. 540–549.

[2] M. R. Brown and R. E. Tarjan, *Design and analysis of a data structure for representing sorted lists*, SIAM J. Comput., 9 (1980), pp. 594–614.

[3] Y. S. Chow and H. Teicher, *Probability Theory. Independence, Interchangeability, Martingales*, Springer-Verlag, New York, Heidelberg, 1978.

[4] L. Devroye, *Applications of the theory of records in the study of random trees*, Acta Inform., 26 (1988), pp. 123–130.

[5] P. Flajolet, T. Ottmann, and D. Wood, *Search trees and bubble memories*, RAIRO Inform. Théor., 19 (1985), pp. 137–164.

[6] S. Huddleston and K. Mehlhorn, *A new data structure for representing sorted lists*, Acta Inform., 17 (1982), pp. 157–184.

[7] G. Louchard, *Exact and asymptotic distributions in digital and binary search trees*, RAIRO Inform. Théor. Appl., 21 (1987), pp. 479–495.

[8] H. M. Mahmoud, *Evolution of Random Search Trees*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Inc., New York, 1992.

[9] H. M. Mahmoud and R. Neininger, *Distribution of distances in random binary search trees*, Ann. Appl. Probab., 13 (2003), pp. 253–276.

[10] H. M. Mahmoud and B. Pittel, *On the most probable shape of a search tree grown from a random permutation*, SIAM J. Algebraic Discrete Methods, 5 (1984), pp. 69–81.

[11] M. Okamoto, *Some inequalities relating to the partial sum of binomial probabilities*, Ann. Inst. Statist. Math., 10 (1958), pp. 29–35.

[12] A. Panholzer and H. Prodinger, *Spanning tree size in random binary search trees*, Ann. Appl. Probab., to appear.

[13] H. Prodinger, *Multiple quickselect—Hoare's find algorithm for several elements*, Inform. Process. Lett., 56 (1995), pp. 123–129.

[14] R. Seidel and C. R. Aragon, *Randomized search trees*, Algorithmica, 16 (1996), pp. 464–497.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 659–673

THE EFFECTS OF TEMPORARY SESSIONS ON NETWORK PERFORMANCE*

MATTHEW ANDREWS† AND LISA ZHANG†

Abstract. We consider a packet network, in which packets are injected in *sessions* along fixed paths. Packet movement is restricted by link bandwidth. In case of contention, a *contention resolution protocol* determines which packets proceed. In the *permanent session model*, a fixed set of connections is present in the network at all times. In the *temporary session model*, connections come and go over time. In this paper we compare network performance in these two models in terms of stability and end-to-end delay.

We provide the first separation of the two models in terms of stability. In particular, we show that generalized processor sharing (GPS) can be unstable with temporary sessions, whereas GPS is known to be stable and have polynomial delay bounds with permanent sessions.

We also observe that the relative performance of protocols can differ in the two models. For example, in the temporary session model the protocol farthest-to-go (FTG) is known to be stable and therefore outperforms GPS. However, in the permanent session model we show that FTG can suffer exponential delays and is therefore outperformed by GPS.

Although polynomial delay bounds are easy to obtain for permanent sessions, this is not the case when sessions can be temporary. We show that a common framework for bounding delays can only lead to superpolynomial bounds in the temporary session model. We also construct superpolynomial lower bounds on delay for a large class of deterministic, distributed protocols that includes the longest-in-system protocol.

Key words. packet networks, scheduling, stability, delay bounds, permanent sessions, temporary sessions

AMS subject classifications. 68M20, 68M10, 68W40

DOI. 10.1137/S0097539702417225

1. Introduction.

Packet switching is a common feature of modern communication networks. In this environment, flows of small packets carry information from one location to another. The movement of packets within the network can be restricted due to limited link bandwidth. If too many packets are contending for the same link, a *contention resolution protocol* is used to determine which packets can proceed and which packets must wait.

Recently, there has been a great deal of work on the analysis of networks that are subject to worst-case traffic patterns. This work can broadly be divided into two groups. In the *session-oriented* model [2, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20], a fixed set of *sessions* (i.e., connections) is present in the network. Each session claims bandwidth along a fixed session path at all times. This model is appropriate for the study of networks where connections persist for long periods. In this paper we refer to it as the *permanent session* model.

Alternatively, in the *temporary session* model sessions come and go over time. A temporary session claims its bandwidth only during its *active* period. The unused link bandwidth during its *inactive* period can be claimed by other sessions. This model is identical to the *adversarial queueing* model of [1, 4] in terms of the permitted injection patterns. However, these two papers have no notion of session rate. We choose to

*Received by the editors November 5, 2002; accepted for publication (in revised form) February 9, 2004; published electronically April 14, 2004.
http://www.siam.org/journals/sicomp/33-3/41722.html
†Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974 (andrews@research.bell-labs.com, ylz@research.bell-labs.com).

perform our analysis in terms of temporary sessions, thereby allowing us to study popular protocols that are rate-based.

Our aim is to discover how the performance of networks with temporary sessions differs from the performance of networks with permanent sessions. To the best of our knowledge, this topic has not been previously studied.

Performance measures. One of the most important measures in network performance is *stability*. A network is *stable* if the total number of packets in the network is bounded by a quantity independent of time. A closely related measure is *end-to-end delay*, i.e., the time taken for a packet to traverse its path. Many interactive applications rely on low delay to function properly.

A necessary condition for stability is that the total bandwidth claimed by all sessions on any link is no more than the link bandwidth. Under this condition, there is more freedom for packet injections in the temporary session model than in the permanent session model. Hence, it is more difficult to achieve stability and small delays when temporary sessions are present.

Protocols. In this paper we focus on simple, deterministic, distributed protocols, such as *shortest-in-system* (SIS), *longest-in-system* (LIS), *farthest-to-go* (FTG), and *generalized processor sharing* (GPS). SIS (resp., LIS) gives priority to the packet that has been in the system the shortest (resp., longest) amount of time, and FTG gives priority to the packet that has the most number of links still to traverse. SIS, LIS, and FTG were all shown to be stable for temporary sessions in [1]. GPS divides the link bandwidth among sessions in proportion to their session rates. (Note that GPS is not well-defined without session rates.) Parekh and Gallager [14, 15] showed that GPS achieves low end-to-end delay in the permanent session model. This work has created a lot of interest in GPS and has often been studied in connection with the provision of quality-of-service in the Internet [17]. We also consider a large class of deterministic, distributed protocols that we call *vulnerable* protocols.

Our results. Network performance can indeed differ depending on whether or not temporary sessions are present. We show in this paper that achieving stability and low delay is strictly more difficult in the temporary session model.

- In section 3, we show that GPS can be unstable in the temporary session model. Although GPS achieves low delays for permanent sessions, our result indicates that its performance can be extremely poor if there are temporary sessions. This is the first result to show that a natural protocol can have different stability properties in the two models;[1] i.e., we have a *separation* in terms of stability.

  We also present extra conditions under which GPS is in fact stable for temporary sessions. However, this stability comes at the expense of a lower link utilization.
- In section 4 we observe that the relative performance of protocols can depend on whether or not temporary sessions are present. In the temporary session model, our previous result indicates that FTG outperforms GPS since FTG is known to be stable. In contrast, in the permanent session model we show that the delays produced by FTG can be $e^{\Omega(d)}$, whereas GPS is known to have polynomial delays. Hence, GPS outperforms FTG.
- In section 5 we give limits on the delay bounds that can be derived in the

[1] One could artificially create a protocol that behaves like the unstable protocol first-in-first-out (FIFO) in the temporary session model and behaves like the stable protocol LIS in the permanent session model. However, such unnatural protocols are not the subject of interest here.

temporary session model. In particular, we examine an open question posed in [1]. Is there a deterministic, distributed protocol that achieves polynomial delays?

We first show that one common technique for bounding delay can only lead to superpolynomial bounds of the form $e^{\Omega(\sqrt{d})}$, where d is the maximum path length of a session. We then construct a lower bound of $e^{\Omega(d)}$ on the actual delays produced by LIS. (Whether or not LIS guarantees polynomial delays was previously unknown.) Finally, we extend the analysis to give a lower bound of $e^{\Omega(\sqrt{d})}$ for a large class of deterministic, distributed protocols that we call *vulnerable* protocols.

However, for permanent sessions simple, deterministic, distributed protocols such as GPS do have polynomial delays.

2. The models. We consider a packet-switched network. Packets are injected into the network over time and are routed along predetermined paths of maximum length d. Packet movement is constrained by link bandwidth. In this paper we assume unit packet sizes and unit link bandwidth; i.e., at each time step one packet is allowed to traverse each link.

The injection of each *session* i is characterized by its injection rate ρ_i and burst size σ_i. In addition each session i is associated with a set of *active periods*, $[a_i^0, b_i^0]$, $[a_i^1, b_i^1], \ldots$, which are time intervals during which session i is allowed to inject packets. We say that a session is active during its active periods and inactive at all other times. In the *permanent sessions* model, every session has active period $[0, \infty)$. In the *temporary sessions* model some sessions can be inactive at times.

The injections into session i are (σ_i, ρ_i)-regulated [6, 7]. That is, the total size of packets injected into session i in any active interval of length t is at most $\sigma_i + \rho_i t$. We must ensure that no link is overloaded, otherwise it is impossible to provide any meaningful performance guarantees. In the permanent sessions model we require that

$$\sum_{i \in S_e} \rho_i \leq 1 \quad \text{for all links } e, \tag{1}$$

where S_e is the set of sessions passing through e. We refer to (1) as the *admissibility condition for permanent sessions*. In the temporary sessions model, we could have an analogous condition to (1) on the ρ_i, namely, $\sum_{i \in S_e(t)} \rho_i \leq 1$, where $S_e(t)$ is the set of *active* sessions passing through e. However, this is not sufficient since session i could become active for a very short instant but still inject packets of size σ_i. Instead we require that, during any time interval of length t, the total number of packets that are allowed to be injected into the network to pass through any link e be at most

$$W + (1 - \varepsilon)t \tag{2}$$

for some W and $\varepsilon \geq 0$. We refer to the above as the *admissibility condition for temporary sessions*.

Note that during its active period a session i may inject packets at a rate lower than ρ_i, or even inject no packets at all. A permanent session is active at all times no matter how few packets it injects, and it can be viewed as reserving the bandwidth of ρ_i along the links of its path at all times. On the contrary, when a temporary session is inactive, its bandwidth can be claimed by other sessions. This difference allows more freedom in admissible packet injections in the temporary session model than in the permanent session model.

We now illustrate a set of injections that are admissible in the temporary session model but not in the permanent sessions model. Consider a network of three links e_1, e_2, and e_3. For even i, during time interval $[2^i, 2^{i+1})$ session 0 injects packets at rate $2/3$ along paths e_1 and e_2; for odd i, during time interval $[2^i, 2^{i+1})$ session 1 injects packets at rate $2/3$ along paths e_1 and e_3. Note that these time intervals are disjoint and increase in length. Therefore these sessions are not (σ, ρ)-regulated for any fixed σ if $\rho < 2/3$. Since both sessions pass through link e_1 the injections do not satisfy the admissibility condition (1) for permanent sessions. However, in the temporary session model we define $[2^i, 2^{i+1})$, where i is even to be the active periods for session 0 and $[2^i, 2^{i+1})$, where i is odd to be the active periods for session 1. Then at any given time instant the total injection rate from the active sessions on server e_1 is equal to $2/3$ and so the injections are admissible in the temporary sessions model.

3. Instability of GPS with temporary sessions. GPS is a protocol designed for the *fluid model*, in which link bandwidth can be *instantaneously* shared among backlogged sessions. (A session is backlogged at a link if it has packets queueing at the link.) In particular, each session i has a specified weight Φ_i. If $B_e(t)$ represents the set of backlogged sessions at link e at time t, then e services session $i \in B_e(t)$ at rate

$$\phi_i = \frac{\Phi_i}{\sum_{j \in B_e(t)} \Phi_j};$$

i.e., e gives session i a fraction ϕ_i of its bandwidth.

In this paper, we adopt the most common method of choosing the weights by setting Φ_i to ρ_i. (This is also known as *rate proportional processor sharing*, or RPPS.) In practice, *weighted fair queueing* (WFQ), a packet-by-packet protocol, is often implemented to emulate the fluid behavior of GPS. In particular, WFQ gives priority to the packet that would finish traversing the link first under GPS. A more detailed description of these protocols can be found in [8, 14, 15].

In [14, 15], Parekh and Gallager derived polynomial delay bounds for GPS and WFQ in the permanent session model. In contrast, we show in the following that GPS and WFQ can be unstable when temporary sessions are present. This implies unbounded delays.

3.1. Instability with temporary sessions. We now show that GPS can be unstable in the temporary session model. We use the "double-ring" network from [1] on which the FIFO, last-in-first-out (LIFO), and nearest-to-go protocols were shown to be unstable. However, the injections into the network are somewhat different. There are six links in the network in total. Four of the links, e_0, f_0, e_1, and f_1, form a ring. The other two links, f_0' and f_1', are parallel to f_0 and f_1, respectively. (See Figure 1.)

THEOREM 1. *The double-ring network is unstable under GPS in the temporary session model.*

Proof. We analyze the packet movement in phases. We inductively assume that at the beginning of the current phase, s packets from some session A are waiting to traverse links e_0 and f_0. At the end of the phase there will be strictly more than s packets waiting to traverse links e_1 and f_1. By symmetry, we can repeat the process indefinitely to build up an unbounded number of packets in the network.

The injection pattern during the current phase is as follows. Let the beginning of the phase be time 0. For each session, we specify its active time period during

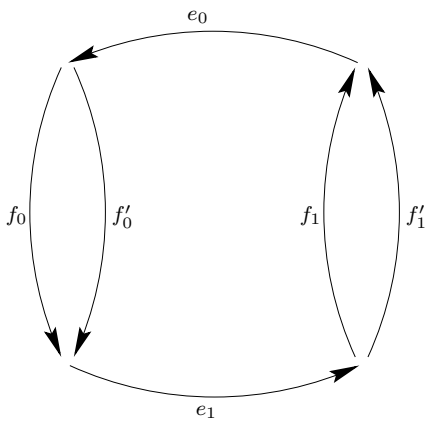


FIG. 1. *The double-ring network.*

which the session injects packets. All sessions have a common injection rate ρ. Our construction creates instability for $\rho \geq 0.973$. Note that the injection pattern satisfies the admissibility condition for temporary sessions (2). Note also that in the permanent session model, the injection is not admissible since, for example, sessions B and D share links e_1 and f_1 and have a total rate higher than 1.

1. During $[0, 2s)$, session B is active on path $e_0 f'_0 e_1 f_1$ and session C is active on path f_0.

2a. During $[2s, 4\rho s)$, session D is active on path $f_0 e_1 f_1$.

2b. During $[2s, 2\rho s + s)$, session E is active on path f'_0.

2c. During $[2\rho s + s, 4\rho s)$, session F is active on path f'_0.

3. During $[4\rho s, 4\rho s + T)$, session G is active on path $e_1 f_1$. We define the parameter T in (3).

At a qualitative level, the above injection pattern creates the following effect. During step 1, session A contends with sessions B and C and slows down their service. As a result, at time $2s$, session B has a backlog at link e_0 and session C has a backlog at link f_0. During step 2a, the backlog of session C slows down the service of session D. During step 2b, session E slows down the service of session B. During step 2c sessions E and F *both* slow down the service of B. At time $4\rho s$, sessions B and D have a total backlog of more than s packets that still have to traverse $e_1 f_1$.

The purpose of the injections in step 3 is to create one single session G with more than s packets waiting to traverse $e_1 f_1$. The state of the network at time $4\rho s + T$ is the same as the state at time 0 except that we now have more packets waiting and they have been shifted from the head of the path $e_0 f_0$ to the head of the path $e_1 f_1$. Hence we can repeat the process indefinitely to create instability.

To obtain a better understanding of the numbers, suppose that ρ is 1. Session G will then have $7s/6$ packets waiting to traverse $e_1 f_1$ at time $4\rho s + T$. For $\rho < 1$, fewer than $7s/6$ packets from session G will be waiting to traverse $e_1 f_1$. However, if we choose ρ sufficiently close to 1, then the number of packets waiting can still be strictly more than s.

Note that it is crucial that E and F are two *different* sessions during step 2c. This creates three sessions B, E, and F that are all competing for link f'_0. Session B is therefore serviced at rate $1/3$. If E and F were the same session, then B would be serviced at rate $1/2$ at f'_0 and at most s packets would be waiting to traverse $e_1 f_1$.

This is not sufficient to create instability.

We now give a more detailed description of the phase. Since all sessions have the same injection rate, under GPS each link services all backlogged sessions at the same rate.

1. During $[0, 2s)$, link e_0 services both sessions A and B at rate $1/2$; link f_0 services sessions A and C at rate $1/2$.
   At time $2s$, there are no session-A packets left in the network; session B has $2s(\rho - 1/2)$ packets queueing at link e_0; session C has $2s(\rho - 1/2)$ packets queueing at link f_0.

2a. During $[2s, 4\rho s)$, link f_0 services sessions C and D at rate $1/2$.
   At time $4\rho s$, session C has no packets left in the network; session D has $(4\rho s - 2s)(\rho - 1/2)$ packets queueing at link f_0.

2b. During $[2s, 2\rho s + s)$, link f_0' services sessions B and E at rate $1/2$; links e_1 and f_1 service sessions B and D at rate $1/2$.
   At time $2\rho s + s$, session B has $2s(\rho - 1/2) - (2\rho s - s)/2 = s(\rho - 1/2)$ packets queueing at link f_0'; session E has $(2\rho s - s)(\rho - 1/2)$ packets queueing at f_0'.

2c. During $[2\rho s + s, 4\rho s)$, *three* sessions B, E, and F are backlogged at link f_0'. Link f_0' services all these sessions at rate $1/3$. Links e_1 and f_1 service session D at rate $1/2$ and they service session B at rate $1/3$. Note that sessions B and D have no backlogs at e_1 and f_1, and hence they are served at the rate at which they arrive.
   At time $4\rho s$, session B has $s(\rho - 1/2) - (2\rho s - s)/3 = s(\rho - 1/2)/3$ packets queueing at link f_0'; sessions E and F both have packets queueing at f_0'.

3. During $[4\rho s, 4\rho s + T)$, link e_1 first services sessions B, D, and G at rate $1/3$ for the first $\rho s - s/2$ steps until session B has no packets left in the network. Link e_1 then services D and G at rate $1/2$ until time $4\rho s + T$, which we define to be the time at which session D has no packets left. The value of T is therefore

$$\begin{aligned} T &= 3 \cdot (s(\rho - 1/2)/3) + 2 \cdot ((4\rho s - 2s)(\rho - 1/2) - s(\rho - 1/2)/3) \\ &= (8\rho s - 11s/3)(\rho - 1/2). \end{aligned} \tag{3}$$

   At time $4\rho s + T$, the number of session G packets that are waiting to traverse e_1 and f_1 is

$$\begin{aligned} &T\rho - s(\rho - 1/2)/3 - (T - s(\rho - 1/2))/2 \\ &= T(\rho - 1/2) + s(\rho - 1/2)/6. \end{aligned}$$

   For $\rho \geq 0.973$, the above quantity is strictly greater than s. □

So far we have considered the GPS protocol that applies to the fluid model in which service can be allocated fractionally among different sessions. The key feature of our instability analysis is that we bound the amount of service that GPS gives to *each session over long time scales*. If we move from the fluid model of GPS to the packetized model of WFQ, we note that the amount of service given to each session by GPS or by WFQ is indistinguishable over large time scales. Therefore we have the following.

COROLLARY 2. *The double-ring network is unstable under WFQ in the temporary session model.*

3.2. Stability at reduced link utilization. By imposing extra conditions, it is possible to ensure network stability under GPS and WFQ when temporary sessions

are present. The instability in the previous section comes from the fact that a new active session can start to inject packets *before* the packets from an inactive session reach their destination. For example, sessions E and F start to inject packets before some session-B packets are at their destination. Let a session be *live* if it has packets that are not yet at their destination. We can choose to not free up the bandwidth used by a live session (even if the session is inactive). In other words, one can enforce the following condition in addition to condition (2).

$$\sum_{i \in L_e(t)} \rho_i \leq 1 - \varepsilon \quad \text{for all links } e \text{ and time } t, \tag{4}$$

where $L_e(t)$ represents the set of sessions that pass through link e and that are live at time t. With this enforcement we can show that GPS and WFQ are stable. However, the total injection rate on some link can be arbitrarily small. The link utilization therefore suffers.

THEOREM 3. *GPS and WFQ are stable in the temporary session model if condition* (4) *holds. However, the total injection rate on some link can be arbitrarily close to* 0.

Proof. If condition (4) holds, then the total injection rate over all live sessions at any link is at most $1 - \varepsilon$ at all time. This allows us to adapt the analysis of Parekh and Gallager [15] to obtain the same polynomial delay bound for GPS and WFQ as in the permanent session model.

Suppose sessions A and B share link e. Both sessions have injection rate $1 - \varepsilon$ and K links on their paths. Suppose A and B alternate between being active and inactive, and the length of their active period is α. The injection rate on link e over time is at most

$$\frac{\alpha(1-\varepsilon)}{\alpha + K},$$

since it takes at least K time steps for the packets to reach their destinations. When K/α is large, the above quantity can be arbitrarily close to 0. ∎

4. Exponential delay of FTG with permanent sessions. In the temporary session model, our result in section 3 indicates that GPS is outperformed by the FTG protocol since FTG is stable [1]. In this section we show that the relative performance of the two protocols is reversed when all sessions are permanent. Parekh and Gallager [15] derived a polynomial delay bound for GPS. We show in the following that the delays under FTG can be of the form $e^{\Omega(d)}$, where d is the maximum path length.

THEOREM 4. *Under FTG, the delays can be* $e^{\Omega(d)}$ *even if all sessions are permanent.*

Proof. We construct a network in which there exists a session A that experiences exponential delay under FTG. Suppose that session A has burst size $\sigma \geq 1$, injection rate $\rho = 0.45$, and path $e_0, e_1, \ldots, e_{d-1}$. At link e_0, A contends with session Z. The path of Z consists of e_0 and then d additional links. Suppose that session Z also has burst size σ.

At time t_0, a burst of σ session-A packets and σ session-Z packets arrives at link e_0. Under FTG, session Z has priority over A on link e_0. Hence, during the time interval $[t_0, t_0 + \sigma)$, the only packets that traverse link e_0 are the σ session-Z packets. During this interval $[t_0, t_0 + \sigma)$, session A injects additional packets at rate ρ.

Session Z injects no packets. Let $t_1 = t_0 + \sigma$. The total number of session-A packets queued at link e_0 at time t_1 is $\sigma(1+\rho)$.

At time $t_{i+1} = t_i + \sigma(1+\rho)^i$, we inductively assume that a total of $\sigma(1+\rho)^{i+1}$ session-A packets are queueing at links $e_0, e_1, \ldots, e_i$, among which at least $\sigma(1+\rho)^i$ are queueing at e_i. In addition we assume no packets from other sessions are queueing at $e_0, \ldots, e_i$ at time t_{i+1}. (As we can see from the previous paragraph, the basis of induction holds for $i = 0$.) Let Π_i be the injection process that created this state and let N_i be the network on which the injections of Π_i are made.

We now create a network N_i' that is identical to N_i except that the end of each session path in N_i' is extended by one extra link. Let session A' in N_i' correspond to session A in N_i, and let $e_0', \ldots, e_i', e_{i+1}$ be the first $i+2$ links of session A'. Networks N_i' and N_i share no common links other than e_{i+1}. (See Figure 2.) The injection process Π_i' in N_i' is identical to Π_i up to time t_{i+1}. After t_{i+1}, session A' injects no packets and session A continues to inject at rate ρ.

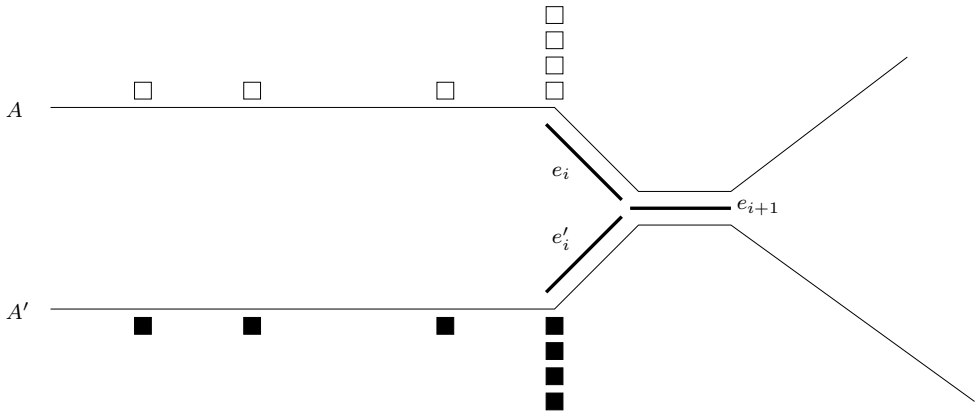


FIG. 2. *Sessions A and A' in the inductive assumption at time t_{i+1}. Other sessions are omitted in this drawing.*

With the above construction, the same inductive assumption holds for network N_i' at time t_{i+1}. Under FTG, session A' has priority over A on link e_{i+1}. During time interval $[t_{i+1}, t_{i+2})$, where $t_{i+2} = t_{i+1} + \sigma(1+\rho)^{i+1}$, $\sigma(1+\rho)^{i+1}$ session-A' packets traverse links e_i' and e_{i+1} one after another. Note that some session-A' packets may not be at e_i' at time t_{i+1}. However, since other sessions have no packets queueing at links $e_0', \ldots, e_i'$, these session-A' packets can reach e_i' by time $t_{i+1} + \sigma(1+\rho)^i$ since $\sigma(1+\rho)^i > i$ for all i when $\rho = 0.45$. By a similar argument, during time interval $[t_{i+1}, t_{i+2})$, $\sigma(1+\rho)^{i+1}$ session-A packets traverse links e_i' one after another and then queue up at e_{i+1}. Meanwhile, $\sigma(1+\rho)^{i+1}\rho$ session-A packets are injected. Thus, a total of $\sigma(1+\rho)^{i+2}$ packets are queueing up at links $e_0, \ldots, e_{i+1}$, among which at least $\sigma(1+\rho)^{i+1}$ are at e_{i+1}. In addition, during $[t_{i+1}, t_{i+2})$ no packets other than those from session A are injected along $e_0, \ldots, e_i$. Since by induction no packets from other sessions are queueing at these links at time t_{i+1}, no packets from other sessions can be queueing at links $e_0, \ldots, e_i$ at t_{i+2}. The inductive step is complete.

We note that ρ is set at 0.45 for two reasons. First, link e_{i+1} is the only link where 2 sessions merge in the inductive step. For $\rho = 0.45$ the admissibility condition of (2) holds for a fixed burst. Second, in the inductive step we need $\sigma(1+\rho)^i > i$ for all i and $\rho = 0.45$ suffices. □

5. Superpolynomial lower bounds on delay for temporary sessions. In this section we derive lower bounds on delay in the temporary session model. In section 5.1 we show that a common framework for deriving delay bounds in this model cannot give bounds that are polynomial in d, the maximum session length. We refer to this framework as the *no-early-arrival* framework. In section 5.2 we extend the result to show that a large class of *deterministic*, *distributed* protocols, including LIS, cannot give polynomial delay bounds. On the other hand protocols such as GPS can guarantee polynomial delays in the permanent session model.

Throughout this section, we focus on the injection of individual packets and do not explicitly define sessions. However, we can regard a single packet injection as a session with a burst size of 1, an arbitrarily small injection rate, and an active period of 1 time step.

5.1. Lower bound for the no-early-arrival framework. We describe the no-early-arrival framework that has been used to derive delay bounds for a number of protocols, e.g., LIS, SIS, and FTG, in [1]. In this framework, for a given protocol $\mathcal{P}$ we create a sequence $T_1, T_2, \ldots$ for which every packet traverses its ith link within T_i steps of its injection. Each T_i is calculated by assuming that each packet traverses its $(i-1)$st link *exactly* at time T_{i-1}. We bound the additional amount of time, S_i, that a packet takes to traverse its ith link. We then set $T_i = T_{i-1} + S_i$. Each S_i is bounded by some inequality expressed in terms of the T's, where this inequality explores the properties of $\mathcal{P}$. If these inequalities have a positive solution, then the delay bounds are obtained.

For example, consider the SIS protocol. Suppose a packet p is injected at time T_0 and arrives at its ith link at time $T_0 + T_{i-1}$. During the time period $[T_0 + T_i, T_0 + T_{i-1} + S_i)$, at most $\sigma + (1-\varepsilon)(T_{i-1} + S_i)$ packets can have priority over packet p at link e. If

$$\begin{aligned} S_i &\geq \frac{\sigma + (1-\varepsilon)T_{i-1}}{\varepsilon} \\ \Rightarrow S_i &\geq \sigma + (1-\varepsilon)(T_{i-1} + S_i), \end{aligned}$$

then packet p traverses its ith link e by time $T_0 + T_{i-1} + S_i$. Thus, we can set $T_i = T_{i-1} + S_i$. By induction we obtain a bound on T_d. (A variant of this proof was used to show stability of SIS in [1].)

The above approach does not take advantage of the fact that some packets may traverse their ith link *earlier* than T_i. We refer to this approach as the *no-early-arrival* framework. In [1], the delay bounds for the three stable protocols LIS, SIS, and FTG were all derived within the no-early-arrival framework. We now show that within this framework we *cannot* prove polynomial delay bounds for *any* protocol. More precisely, we have the following.

THEOREM 5. *If the no-early-arrival framework is used to bound the delay in the temporary session model, then for any protocol* $T_d = e^{\Omega(\sqrt{d})}$.

Proof. Suppose that $T_1, T_2, \ldots, T_d$ is a sequence of delay bounds that has been derived for a protocol $\mathcal{P}$ using the no-early-arrival framework. We use the following injections to obtain a lower bound for T_d. All of the packets are to traverse some common link e. The packets are divided into $d-1$ groups. Let $\tau \geq T_{d-1}$ and $T_0 = 0$. For $1 \leq i \leq d-1$, group i consists of $(1-\varepsilon)(T_i - T_{i-1})$ packets that are injected during time interval $[\tau - T_i, \tau - T_{i-1})$. These packets in group i have link e as the $(i+1)$st link on their paths. (See Figure 3.) Since the different groups inject packets at disjoint time intervals, condition (2) is satisfied. In the no-early-arrival framework,

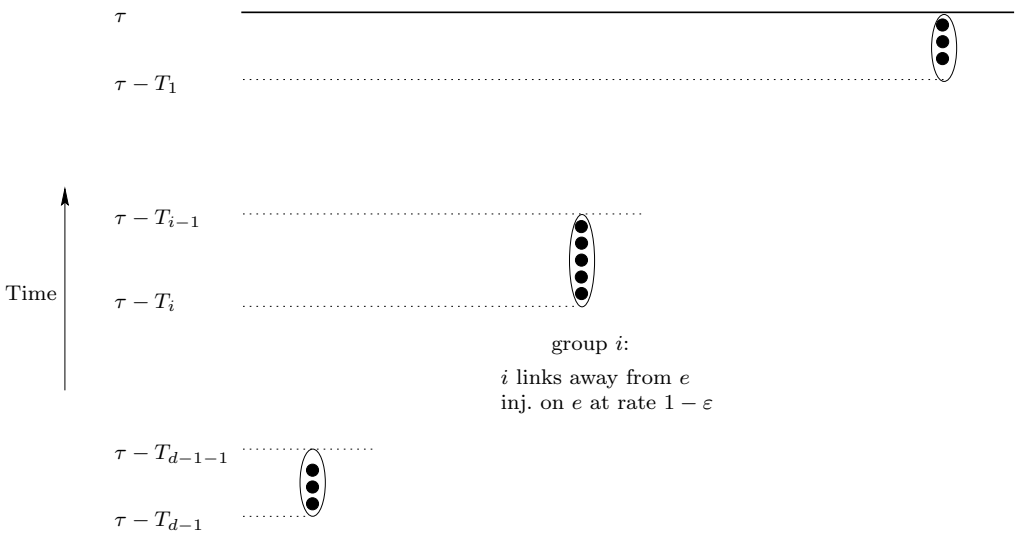


FIG. 3. *Packets arrive at e no earlier than τ in the no-early-arrival framework.*

we assume that all packets from these $d-1$ groups arrive at link e no earlier than time τ.

We order the groups by the time at which all the packets from the group have traversed link e under protocol $\mathcal{P}$. Let $\pi_1, \pi_2, \ldots, \pi_{d-1}$ be this ordering; i.e., if $i < j$, all the packets from π_i traverse link e before the last packet of π_j. Let p be the last packet from group π_i to traverse link e. Since p is in group π_i, it is injected by time $\tau - T_{\pi_i - 1}$ and therefore traverses link e by time $\tau + T_{\pi_i+1} - T_{\pi_i-1}$. By the definition of the ordering, all packets from groups $\pi_1, \ldots, \pi_i$ traverse link e before p. Since all these packets arrive at link e no earlier than τ and $(1-\varepsilon)(T_{\pi_j} - T_{\pi_j - 1})$ packets are in group π_j, we have the following recurrence:

(5)
$$T_{\pi_i+1} - T_{\pi_i-1} \geq (1-\varepsilon)[(T_{\pi_i} - T_{\pi_i-1}) + (T_{\pi_{i-1}} - T_{\pi_{i-1}-1}) + \cdots + (T_{\pi_1} - T_{\pi_1-1})].$$

We proceed to show that $T_d = e^{\Omega(\sqrt{d})}$ regardless of the ordering, π. By letting $S_i = T_i - T_{i-1}$, the recurrence (5) is equivalent to the following:

$$S_{\pi_i+1} + S_{\pi_i} \geq (1-\varepsilon)(S_{\pi_i} + S_{\pi_{i-1}} + \cdots + S_{\pi_1}). \tag{6}$$

For simplicity, we refer to each recurrence (6) by its subscript π_i. We say that a recurrence π_i is *right-big* if $S_{\pi_{i+1}} \leq S_{\pi_i}$; a recurrence is *left-big* otherwise. Note that if recurrence π_i is right-big, then $S_{\pi_i} \geq \frac{1-\varepsilon}{2}(S_{\pi_i} + S_{\pi_{i-1}} + \cdots + S_{\pi_1})$. If π_i is left-big, then $S_{\pi_i+1} \geq \frac{1-\varepsilon}{2}(S_{\pi_i} + S_{\pi_{i-1}} + \cdots + S_{\pi_1})$. We also say that recurrences π_j for $j > i$ are *above* recurrence π_i and that recurrences π_j for $j < i$ are *below* recurrence π_i.

It suffices to show that $S_i = e^{\Omega(\sqrt{d})}$ for some i, since $T_d = \sum_{1 \leq i \leq d} S_i$. We observe that S_{π_i} appears on the right-hand side (RHS) of the recurrence π_i. Also, every term on the RHS of π_i appears on the RHS of all the recurrences above π_i. These two simple observations allow us to prove lower bounds on any subset of *right-big* recurrences.

LEMMA 6. *If $\varphi_1, \ldots, \varphi_k$ are right-big recurrences such that φ_i appears above φ_{i-1} for all $1 < i \leq k$, then $\sum_{1 \leq i \leq k} S_{\varphi_i}$ is at least $\left(1 + \frac{1-\varepsilon}{2}\right)^{k-1} S_{\varphi_1}$. As a result, $S_{\varphi_k} = e^{\Omega(k)}$.*

Proof. The lemma holds trivially for $k = 1$. Inductively, we have

$$\begin{aligned} S_{\varphi_k} &\geq \frac{1-\varepsilon}{2}(S_{\varphi_k} + S_{\varphi_{k-1}} + \cdots + S_{\varphi_1}) \\ &\geq \frac{1-\varepsilon}{2}\left(1 + \frac{1-\varepsilon}{2}\right)^{k-2} S_{\varphi_1}. \end{aligned}$$

The first inequality holds since recurrence φ_k is right-big and $S_{\varphi_k}, \ldots, S_{\varphi_1}$ all appear on the RHS of recurrence φ_k by our earlier observations. The second inequality follows from the inductive hypothesis. ∎

If we have more than $\sqrt{d}$ right-big recurrences, then by Lemma 6 we are done. However, this is not always true. Our general strategy is to prove a lower bound exponential in k, where k is the length of the longest monotone subsequence of $\pi_1, \ldots, \pi_{d-1}$. It is well known that $k \geq \sqrt{d-1}$. Let $\varphi_1, \ldots, \varphi_k$ be a longest monotone subsequence.

We define a sequence of *delay lower bounds*, B_i, such that the RHS of recurrence φ_i is at least $(1-\varepsilon)B_i$. For the base case, let $B_1 = S_{\varphi_1}$. The RHS of recurrence φ_1 is at least $(1-\varepsilon)B_1$. For recurrence φ_i where $i \geq 2$, we either show that its RHS is at least $(1-\varepsilon)\left(1 + \frac{1-\varepsilon}{2}\right) B_{i-1}$ or else we find a new right-big recurrence. In the former case, we set $B_i = (1 + \frac{1-\varepsilon}{2})B_{i-1}$. In the latter case we set $B_i = B_{i-1}$. Hence, either at least $(k-1)/2$ recurrences are right-big or else $B_k \geq \left(1 + \frac{1-\varepsilon}{2}\right)^{(k-1)/2} S_{\varphi_1}$. In the former case we have $S_{\varphi_k} = e^{\Omega(k)}$ by Lemma 6. In the latter case we have $S_{\varphi_k+1} + S_{\varphi_k} \geq (1-\varepsilon)B_k = e^{\Omega(k)}$. In both cases we have

$$T_d = \sum_{1 \leq i \leq d} S_i \geq S_{\varphi_k+1} + S_{\varphi_k} = e^{\Omega(\sqrt{d})}.$$

The details of the analysis depend on whether or not the monotone subsequence is increasing or decreasing.

Case 1: *Increasing subsequence,* $\varphi_1 < \cdots < \varphi_k$. For each recurrence φ_i where $2 \leq i \leq k$ we define a *helper* as follows. If recurrence $\varphi_{i-1} + 1$ is below or the same as φ_i, then recurrence φ_{i-1} is the helper. Otherwise, there exists some j where $\varphi_{i-1} < j < \varphi_i$ such that recurrence j is above φ_i and recurrence $j+1$ is below or the same as φ_i. We define recurrence j to be the helper. Suppose recurrence j is the helper of φ_i. We observe the following.

PROPOSITION 7. *The term S_{j+1} is on the RHS of φ_i and all the recurrences above φ_i.*

PROPOSITION 8. *Recurrence j is no lower than recurrence φ_{i-1}.*

If j is left-big, we can lower bound S_{j+1} using the RHS of φ_{i-1} and then lower bound the RHS of φ_i using S_{j+1}. More precisely, let us define $W_{\varphi_i} = W_{\varphi_{i-1}} + S_{j+1}$ and $W_{\varphi_1} = S_{\varphi_1}$. Since the RHS of recurrence φ_i is at least $(1-\varepsilon)W_{\varphi_i}$ due to Proposition 7, it suffices to lower bound W_{φ_i}. Assume inductively that $W_{\varphi_{i-1}} \geq B_{i-1}$. We now consider recurrence φ_i and let j be its helper. If j is left-big, we have

$$S_{j+1} \geq \frac{1}{2} \cdot \text{RHS of recurrence } \varphi_{i-1} \geq \frac{1-\varepsilon}{2} W_{\varphi_{i-1}}.$$

The first inequality holds because j is left-big. The second inequality follows from Proposition 8. As a result,

$$\begin{aligned} W_{\varphi_i} &= S_{j+1} + W_{\varphi_{i-1}} \\ &\geq \left(1 + \frac{1-\varepsilon}{2}\right) W_{\varphi_{i-1}} \geq \left(1 + \frac{1-\varepsilon}{2}\right) B_{i-1}. \end{aligned}$$

Hence, we set $B_i = \left(1 + \frac{1-\varepsilon}{2}\right) B_{i-1}$. If recurrence j is right-big, then we have found a new right-big recurrence. Note that j is distinct from any right-big recurrence found earlier since our construction ensures that all helpers are distinct. Since φ_i is above φ_{i-1} the RHS of φ_i is at least as big as the RHS of φ_{i-1}. Hence, we set $B_i = B_{i-1}$.

At the conclusion of this process, either $B_k \geq \left(1 + \frac{1-\varepsilon}{2}\right)^{(k-1)/2} S_{\varphi_1}$ or we have at least $(k-1)/2$ distinct right-big recurrences. As described earlier, in both cases we have $T_d = e^{\Omega(\sqrt{d})}$.

Case 2: *Decreasing subsequence,* $\varphi_1 > \cdots > \varphi_k$. The analysis for a decreasing subsequence is similar. We highlight the differences here. For each recurrence φ_i where $2 \leq i \leq k$ we define its helper as follows. If $\varphi_i + 1$ is below recurrence φ_i, then φ_i is its own helper. Otherwise, there exists some j where $\varphi_i < j < \varphi_{i-1}$ such that recurrence j is above φ_i and recurrence $j+1$ is below φ_i. We define recurrence j to be the helper. As before, if j is the helper of φ_i, then it satisfies Propositions 7 and 8. If j is left-big, we lower bound S_{j+1} and this in turn lower bounds the RHS of φ_i. Otherwise, j is right-big. We handle the set of right-big recurrences using Lemma 6. □

The analysis of Theorem 5 shows that the exponent in the lower bound is determined by the length of the longest monotone sequence. The sequence $\pi_1, \ldots, \pi_{d-1}$ is increasing for protocol SIS and is decreasing for protocol LIS. Hence, we have the following.

COROLLARY 9. *If the no-early-arrival framework is used to bound the delay in the temporary session model, then* $T_d = e^{\Omega(d)}$ *for SIS and LIS.*

5.2. Lower bounds for a class of deterministic, distributed protocols. In the previous section we concentrated on a particular technique for deriving delay bounds. In this section we show that the above arguments can be used to lower bound the actual delay produced by many protocols. It was proven in [1] that the delays due to SIS and FTG can be $e^{\Omega(d)}$. However, the question of whether or not LIS has polynomial delays was previously unresolved. More generally, [1] poses the question of whether or not any deterministic, distributed protocol can guarantee polynomial delays. In the following, we construct an exponential bound $e^{\Omega(d)}$ for the actual delays produced by LIS. We then extend the argument to show a superpolynomial lower bound for a large class of deterministic, distributed protocols.

THEOREM 10. *Under LIS, there exists some network for which some packet takes* $e^{\Omega(d)}$ *steps to traverse a path of* d *links in the temporary session model.*

Proof. We construct the network in stages. At stage j we inductively obtain a set of networks $N_1^{(j)}, \ldots, N_{d-1}^{(j)}$ and a set of delay bounds $T_1^{(j)} < T_2^{(j)} < \cdots < T_{d-1}^{(j)}$ such that under some injection process $\Pi_i^{(j)}$ some packet p_i takes exactly $T_i^{(j)}$ time steps to traverse an i-link path P_i in network $N_i^{(j)}$. At stage 1 we have $T_i^{(1)} = i$ and the network $N_i^{(1)}$ is a linear array of i links. The injection process $\Pi_i^{(1)}$ injects one packet that must traverse all i links of $N_i^{(1)}$.

Given the set of networks at stage j, we now show how to construct the networks at stage $j+1$. The idea of our analysis is similar to that in Theorem 5. We create one big network N which consists of one new link e and $(1-\varepsilon)(T_i^{(j)} - T_{i-1}^{(j)})$ copies of network $N_i^{(j)}$ for $1 \leq i \leq d-1$. The link e is appended to the tail of the path P_i for every copy of $N_i^{(j)}$. (See Figure 4.) We modify the injection process $\Pi_i^{(j)}$ so that p_i is to traverse link e after traversing path P_i. It is important to observe the following.

PROPOSITION 11. *Under LIS, the packet movement within* $N_i^{(j)}$ *due to the*

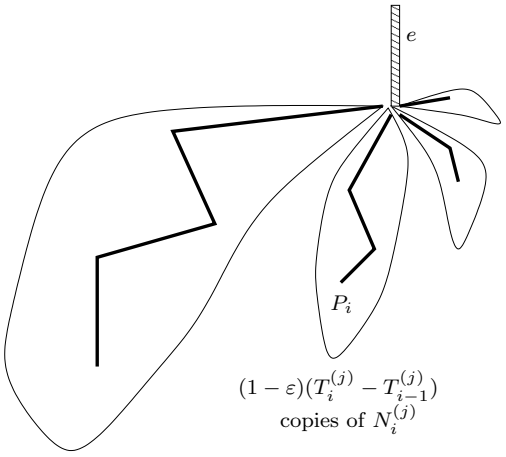


FIG. 4. *The construction of network N from $N_1^{(j)}, \ldots, N_d^{(j)}$.*

injection process $\Pi_i^{(j)}$ is unchanged by these modifications.

The injection process Π on N is as follows. Let $\tau \geq T_{d-1}^{(j)}$. We space out the starting time of the injection process $\Pi_i^{(j)}$ on each copy of $N_i^{(j)}$. In particular, the ℓth copy, where $0 \leq \ell < (1-\varepsilon)(T_i^{(j)} - T_{i-1}^{(j)})$, starts at time $\tau - T_i^{(j)} + \ell/(1-\varepsilon)$. This ensures that the p_i's are injected at the new link e at a rate $1-\varepsilon$. (These packets injected by the processes $\Pi_i^{(j)}$ are equivalent to the packets in group i in Theorem 5. See Figure 3.) We note that the injection process Π on N is admissible by (2) since the injection process $\Pi_i^{(j)}$ is admissible on each copy of $N_i^{(j)}$ and the new link e is not overloaded due to the spacing of the starting times of $\Pi_i^{(j)}$. There are two cases to consider.

Case 1. Under LIS, every packet p_i for $1 \leq i \leq d-2$ traverses link e within $T_{i+1}^{(j)}$ steps of its injection. By induction, packet p_i arrives at the head of link e exactly $T_i^{(j)}$ steps after its injection. Hence recurrences (5) hold for a decreasing sequence $\pi_1, \ldots, \pi_{d-1}$. In particular,

$$T_i^{(j)} - T_{i-2}^{(j)} \geq (1-\varepsilon)(T_{d-1}^{(j)} - T_{i-2}^{(j)}) \quad \text{for } 2 \leq i \leq d-1.$$

This inequality is equivalent to (5). By an argument similar to the proof of Theorem 5, we have $T_{d-1}^{(j)} = e^{\Omega(d)}$.

Case 2. Under LIS, some packet p_i traverses link e at a time T' steps after its injection where $T' > T_{i+1}^{(j)}$. Therefore, we can set $N_{i+1}^{(j+1)} = N$, $T_{i+1}^{(j+1)} = T'$, and $\Pi_{i+1}^{(j+1)} = \Pi$. The new path P_{i+1} is the old path P_i concatenated with link e. For $i' \neq i+1$, it is trivial to construct networks and injection processes for which we can set the $T_{i'}^{(j+1)}$ so that $T_{i'}^{(j+1)} \geq T_{i'}^{(j)}$ and $T_{i'}^{(j+1)} > T_{i'-1}^{(j+1)}$.

Note that for Case 2, one of the T_i values *strictly* increases. Hence after $j = de^{\Omega(d)}$ stages we must have $T_i^{(j)} = e^{\Omega(d)}$ for some i. □

For any protocol, if Proposition 11 holds, then we can construct a network in stages as shown in Theorem 10. More precisely, we say that protocol P is *vulnerable* if

1. protocol P is deterministic;

2. protocol P is independent of the remainder of the path that each packet still has to traverse;
3. given isomorphic networks A and B with identical injection processes, the behavior of protocol P is identical in A and B.

Note that for an arbitrary vulnerable protocol we can have an arbitrary ordering of the recurrences (5). Hence, the analysis gives a lower bound that is exponential in $\sqrt{d}$.

COROLLARY 12. *For a vulnerable protocol P, there exists some network for which some packet takes $e^{\Omega(\sqrt{d})}$ steps to traverse a path of d links in the temporary session model.*

We remark that SIS, LIS, and GPS are all vulnerable protocols. In addition, by adapting the construction of the networks N we can obtain a lower bound of $e^{\Omega(d)}$ for FTG even though it is not strictly vulnerable. (FTG is dependent on the remainder of a packet's path.) We also remark that our lower bound does not apply to randomized protocols or centralized protocols.

6. Conclusion. In this paper we have shown that the presence of temporary sessions can cause significant deterioration in network performance. There are a number of open problems. Note that for the lower bounds in sections 4 and 5.2 the network sizes are exponential in d. Although we believe that d is the correct parameter to consider when deriving delay bounds, it would be interesting to know if our arguments can be adapted to give lower bounds that are superpolynomial in network size.

For permanent sessions, protocols with near-optimal delay performance were described in [2, 13, 16]. It would be interesting to know if these ideas can be applied to temporary sessions. In particular, it is not clear how to adapt these protocols so that they are stable for the network in section 3. We note that if the link utilization is sufficiently small, the protocols of [3] provide low delay bounds in the temporary session model.

REFERENCES

[1] M. ANDREWS, B. AWERBUCH, A. FERNÁNDEZ, J. KLEINBERG, T. LEIGHTON, AND Z. LIU, *Universal stability results and performance bounds for greedy contention-resolution protocols*, J. ACM, 48 (2001), pp. 39–69.

[2] M. ANDREWS, A. FERNÁNDEZ, M. HARCHOL-BALTER, T. LEIGHTON, AND L. ZHANG, *General dynamic routing with per-packet delay guarantees of $o(distance + 1/session\ rate)$*, SIAM J. Comput., 30 (2000), pp. 1594–1623.

[3] P. BERENBRINK AND C. SCHEIDELER, *Locally efficient on-line strategies for routing packets along fixed paths*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, MD, 1999), ACM, New York, SIAM, Philadelphia, pp. 112–121.

[4] A. BORODIN, J. KLEINBERG, P. RAGHAVAN, M. SUDAN, AND D. P. WILLIAMSON, *Adversarial queueing theory*, J. ACM, 48 (2001), pp. 13–38.

[5] D. CLARK, S. SHENKER, AND L. ZHANG, *Supporting real-time applications in an integrated services packet network: Architecture and mechanism*, in Proceedings of ACM SIGCOMM '92, ACM, New York, 1992, pp. 14–26.

[6] R. L. CRUZ, *A calculus for network delay, part* I: *Network elements in isolation*, IEEE Trans. Inform. Theory, 37 (1991), pp. 114–131.

[7] R. L. CRUZ, *A calculus for network delay, part* II: *Network analysis*, IEEE Trans. Inform. Theory, 37 (1991), pp. 132–141.

[8] A. DEMERS, S. KESHAV, AND S. SHENKER, *Analysis and simulation of a fair queueing algorithm*, J. Internetworking: Research and Experience, 1 (1990), pp. 3–26.

[9] A. ELWALID AND D. MITRA, *Design of generalized processor sharing schedulers which statistically multiplex heterogeneous QoS classes*, in Proceedings of IEEE INFOCOM '99, IEEE, New York, 1999, pp. 1220–1230.

[10] L. Georgiadis, R. Guérin, and A. Parekh, *Optimal multiplexing on a single link: Delay and buffer requirements*, IEEE Trans. Inform. Theory, 43 (1997), pp. 1518–1535.

[11] L. Georgiadis, R. Guérin, V. Peris, and K. Sivarajan, *Efficient network QoS provisioning based on per node traffic shaping*, in Proceedings of IEEE INFOCOM '96, IEEE, New York, 1996, pp. 102–110.

[12] J. Liebeherr, D. Wrege, and D. Ferrari, *Exact admission control for networks with a bounded delay service*, IEEE/ACM Trans. Networking, 4 (1996), pp. 885–901.

[13] R. Ostrovsky and Y. Rabani, *Universal $O(congestion + dilation + \log^{1+\epsilon} n)$ local control packet switching algorithm*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 644–653.

[14] A. K. Parekh and R. G. Gallager, *A generalized processor sharing approach to flow control in integrated services networks: The single-node case*, IEEE/ACM Trans. Networking, 1 (1993), pp. 344–357.

[15] A. K. Parekh and R. G. Gallager, *A generalized processor sharing approach to flow control in integrated services networks: The multiple-node case*, IEEE/ACM Trans. Networking, 2 (1994), pp. 137–150.

[16] Y. Rabani and É. Tardos, *Distributed packet switching in arbitrary networks*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing (Philadelphia, PA, 1996), ACM, New York, pp. 366–375.

[17] S. Shenker, C. Partridge, and R. Guérin, *Specification of Guaranteed Quality of Service*, RFC 2212, IETF, 1997; www.ietf.org.

[18] D. Stiliadis, *Traffic Scheduling in Packet-Switched Networks: Analysis Design and Implementation*, Ph.D. thesis, UCSC, Santa Cruz, CA, 1996.

[19] D. Wrege and J. Liebeherr, *A near-optimal packet scheduler for QoS networks*, in Proceedings of IEEE INFOCOM '97, IEEE, New York, 1997.

[20] H. Zhang, *Service disciplines for guaranteed performance service in packet-switching networks*, Proc. IEEE, 83 (1995), pp. 1374–1396.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 674–703

COMPLETE AXIOMATIZATIONS FOR REASONING ABOUT KNOWLEDGE AND TIME*

JOSEPH Y. HALPERN†, RON VAN DER MEYDEN‡, AND MOSHE Y. VARDI§

Abstract. Sound and complete axiomatizations are provided for a number of different logics involving modalities for knowledge and time. These logics arise from different choices for various parameters regarding the interaction of knowledge with time and regarding the language used. All the logics considered involve the discrete time linear temporal logic operators "next" and "until" and an operator for the knowledge of each of a number of agents. Both the single-agent and multiple-agent cases are studied: in some instances of the latter there is also an operator for the common knowledge of the group of all agents. Four different semantic properties of agents are considered: whether they (i) have a unique initial state, (ii) operate synchronously, (iii) have perfect recall, and (iv) learn. The property of no learning is essentially dual to perfect recall. Not all settings of these parameters lead to recursively axiomatizable logics, but sound and complete axiomatizations are presented for all the ones that do.

Key words. complete axiomatizations, knowledge, common knowledge, temporal logic, epistemic logic

AMS subject classifications. 03B42, 03B70, 68T30

DOI. 10.1137/S0097539797320906

1. Introduction.

It has recently been argued that knowledge is a useful tool for analyzing the behavior and interaction of agents in a distributed system (see [1] and the references therein). When analyzing a system in terms of knowledge, not only is the current state of knowledge of the agents in the system relevant, but it is also relevant how that state of knowledge changes over time. A formal propositional logic of knowledge and time was first proposed by Sato [17]; many others have since been proposed [2, 12, 11, 15, 18]. Unfortunately, while these logics often use similar or identical notation, they differ in a number of significant respects.

In [8], logics for knowledge and time were categorized along two major dimensions: the language used and the assumptions made on the underlying distributed system. The properties of knowledge in a system turn out to depend in subtle ways on these assumptions. The assumptions considered in [8] concern whether agents have *unique initial states*, operate *synchronously* or *asynchronously*, have *perfect recall*, and whether they satisfy a condition called *no learning*. There are 16 possible combinations of these assumptions on the underlying system. Together with 6 choices of language, this gives us 96 logics in all. All the logics considered in the papers

*Received by the editors April 17, 1997; accepted for publication (in revised form) February 9, 2004; published electronically April 14, 2004. This paper incorporates results from [5, 7, 14].
http://www.siam.org/journals/sicomp/33-3/32090.html

†Computer Science Department, Cornell University, Ithaca, NY 14853 (halpern@cs.cornell.edu, http://www.cs.cornell.edu/home/halpern). Much of the work on this paper was carried out while this author was at the IBM Almaden Research Center. IBM's support is gratefully acknowledged. The work was also supported in part by the NSF, under grants IRI-95-03109 and IRI-96-25901, and the Air Force Office of Scientific Research (AFSC), under grant F94620-96-1-0323.

‡School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia, and National ICT Australia (meyden@cse.unsw.edu.au, http://www.cse.unsw.edu.au/∼meyden). The work of this author was supported by an ARC Research Council Large Grant. National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

§Computer Science Department, Rice University, Houston, TX 77005-1892 (vardi@cs.rice.edu, http://www.cs.rice.edu/∼vardi).

mentioned above fit into the framework. In [6, 7, 8], the computational complexity of these logics is completely characterized; the results of these papers show how the subtle interplay of the parameters can have a tremendous impact on complexity. The complexity results show that some of these logics cannot be given a recursive axiomatization, since the set of valid formulas for these logics is not recursively enumerable (r.e.). Of these 96 logics, 48 involve *linear time* and 48 involve *branching time*. (The distinction between linear and branching time essentially amounts to whether or not we can quantify over the possible executions of a program [16].) To keep this paper to manageable length, we focus here on the linear time logics, and provide axiomatic characterizations of all the linear time logics for which an axiomatization is possible at all (i.e., for those logics for which the set of valid formulas is r.e.).

The rest of this paper is organized as follows. In the next section, we provide formal definitions for the logics we consider. In section 2, we review the syntax and semantics of all the logics of knowledge and time that we consider here. In particular, we review the four assumptions on the underlying system that we axiomatize in this paper. In section 3, we state the axioms for all the systems. In section 4, we introduce the notion of *enriched systems*, which form the basis for all our completeness proofs. In section 5, we prove soundness and completeness for the axiom systems described in section 3. The definition of no learning that we use here is slightly different from that used in [1, 5], although they agree in many cases of interest. We discuss the motivation for our change in section 6. We conclude with some further discussion in section 7.

2. The formal model: Language and systems. The material in this section is largely taken from [8] and is repeated here to make this paper self-contained. The reader is encouraged to consult [8] for further details and motivation.

The logics we are considering are all propositional. Thus, we start out with primitive propositions $p, q, \ldots$ and we close the logics under negation and conjunction, so that if φ and ψ are formulas, so are $\neg\varphi$ and $\varphi \wedge \psi$. In addition, we close off under modalities for knowledge and time, as discussed below. As usual, we view *true* as an abbreviation for $\neg(p \wedge \neg p)$, $\varphi \vee \psi$ as an abbreviation for $\neg(\neg\varphi \wedge \neg\psi)$, and $\varphi \Rightarrow \psi$ as an abbreviation for $\neg\varphi \vee \psi$.

If we have m agents (in distributed systems applications, this would mean a system with m processors), we add the modalities $K_1, \ldots, K_m$. Thus, if φ is a formula, so is $K_i\varphi$ (read "agent i knows φ"). We take $L_i\varphi$ to be an abbreviation for $\neg K_i \neg\varphi$. In some cases we also want to talk about common knowledge, so we add the modalities E and C into the language; $E\varphi$ says that everyone knows φ, while $C\varphi$ says φ is common knowledge.

There are two basic linear temporal modalities (sometimes called *operators* or *connectives*): a unary operator $\bigcirc$ and a binary operator U. Thus, if φ and ψ are formulas, then so are $\bigcirc\varphi$ (read "next time φ") and $\varphi\, U\, \psi$ (read "φ until ψ"). $\Diamond\varphi$ is an abbreviation for $true\, U\, \varphi$, while $\Box\varphi$ is an abbreviation for $\neg\Diamond\neg\varphi$. Intuitively, $\bigcirc\varphi$ says that φ is true at the next point (one time unit later), $\varphi U \psi$ says that φ holds until ψ does, $\Diamond\varphi$ says that φ is eventually true (either in the present or at some point in the future), and $\Box\varphi$ says that φ is always true (in the present and at all points in the future). In [8], branching time operators are also considered, which have quantifiers over runs. For example, $\forall\bigcirc$ is a branching time operator such that $\forall\bigcirc\varphi$ is true when $\bigcirc\varphi$ is true for all possible futures. Since we do not consider branching time operators in this paper, we omit the formal definition here. We take CKL_m to be the language for m agents with all the modal operators for knowledge and linear time discussed above; KL_m is the restricted version without the E and C operators.

A *system* for m agents consists of a set $\mathcal{R}$ of runs, where each run $r \in \mathcal{R}$ is a function from $\mathbb{N}$ to $\mathbf{L}^{m+1}$, where $\mathbf{L}$ is some set of *local states*. There is a local state for each agent, together with a local state for the *environment*; intuitively, the environment keeps track of all the relevant features of the system not described by the agents' local states, such as messages in transit but not yet delivered. Thus, $r(n)$ has the form $\langle l_e, l_1, \ldots, l_m \rangle$, where l_e is the state of the environment, and l_i is the local state of agent i for $i = 1, \ldots, m$; such a tuple is called a *global state*. (Formally, we could view a system as a tuple $(\mathcal{R}, \mathbf{L}, m)$, making the $\mathbf{L}$ and m explicit. We have chosen not to do so in order to simplify notation. The $\mathbf{L}$ and m should always be clear from context.) An *interpreted system $\mathcal{I}$ for m agents* is a tuple $(\mathcal{R}, \pi)$, where $\mathcal{R}$ is a system for m agents, and π maps every point $(r, n) \in \mathcal{R} \times \mathbb{N}$ to a truth assignment $\pi(r, n)$ to the primitive propositions (so that $\pi(r, n)(p) \in \{\mathbf{true}, \mathbf{false}\}$ for each primitive proposition p).[1]

We now give semantics to CKL_m and KL_m. Given an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$, we write $(\mathcal{I}, r, n) \vDash \varphi$ if the formula φ is true at (or *satisfied by*) the point (r, n) of interpreted system $\mathcal{I}$. We define $\vDash$ inductively for formulas of CKL_m (for KL_m we just omit the clauses involving C and E). In order to give the semantics for formulas of the form $K_i\varphi$, we need to introduce one new notion. If $r(n) = \langle l_e, l_1, \ldots, l_m \rangle$, $r'(n') = \langle l'_e, l'_1, \ldots, l'_m \rangle$, and $l_i = l'_i$, then we say that $r(n)$ and $r'(n')$ are *indistinguishable to agent i* and write $(r, n) \sim_i (r', n')$. Of course, $\sim_i$ is an equivalence relation on global states (inducing an equivalence relations on points). $K_i\varphi$ is defined to be true at (r, n) exactly if φ is true at all the points whose associated global state is indistinguishable to i from that of (r, n). We proceed as follows:

- $(\mathcal{I}, r, n) \vDash p$ for a primitive proposition p iff $\pi(r, n)(p) = \mathbf{true}$.
- $(\mathcal{I}, r, n) \vDash \varphi \wedge \psi$ iff $(\mathcal{I}, r, n) \vDash \varphi$ and $(\mathcal{I}, r, n) \vDash \psi$.
- $(\mathcal{I}, r, n) \vDash \neg\varphi$ iff $(\mathcal{I}, r, n) \nvDash \varphi$.
- $(\mathcal{I}, r, n) \vDash K_i\varphi$ iff $(\mathcal{I}, r', n') \vDash \varphi$ for all (r', n') such that $(r, n) \sim_i (r', n')$.
- $(\mathcal{I}, r, n) \vDash E\varphi$ iff $(\mathcal{I}, r', n') \vDash K_i\varphi$ for $i = 1, \ldots, m$.
- $(\mathcal{I}, r, n) \vDash C\varphi$ iff $(\mathcal{I}, r', n') \vDash E^k\varphi$, for $k = 1, 2, \ldots$ (where $E^1\varphi = E\varphi$ and $E^{k+1}\varphi = EE^k\varphi$).
- $(\mathcal{I}, r, n) \vDash \bigcirc\varphi$ iff $(\mathcal{I}, r, n + 1) \vDash \varphi$.
- $(\mathcal{I}, r, n) \vDash \varphi U \psi$ iff there is some $n' \geq n$ such that $(\mathcal{I}, r, n') \vDash \psi$, and for all n'' with $n \leq n'' < n'$, we have $(\mathcal{I}, r, n'') \vDash \varphi$.

There is a graphical interpretation of the semantics of C which we shall find useful in what follows. Fix an interpreted system $\mathcal{I}$. A point (r', n') in $\mathcal{I}$ is *reachable* from a point (r, n) if there exist points $(r_0, n_0), \ldots, (r_k, n_k)$ such that $(r, n) = (r_0, n_0)$, $(r', n') = (r_k, n_k)$, and for all $j = 0, \ldots, k - 1$ there exists i such that $(r_j, n_j) \sim_i (r_{j+1}, n_{j+1})$. The following result is well known (and easy to check).

LEMMA 2.1 (see [4]). *$(\mathcal{I}, r, n) \vDash C\varphi$ iff $(\mathcal{I}, r', n') \vDash \varphi$ for all points (r', n') reachable from (r, n).*

As usual, we define a formula φ to be *valid with respect to a class $\mathcal{C}$ of interpreted systems* iff $(\mathcal{I}, r, n) \vDash \varphi$ for all interpreted systems $\mathcal{I} \in \mathcal{C}$ and points (r, n) in $\mathcal{I}$. A

[1] Note that while we are being consistent with [8] here, in [1], π is taken to be a function from global states (not points) to truth values. Essentially, this means that in [1] a more restricted class of structures is considered, where π is forced to be the same at any two points associated with the same global state. Clearly our soundness results hold in the more restricted class of structures. It is also easy to see that our completeness results hold in the more restricted class too. All our completeness proofs have (or can be easily modified to have) the property that a structure is constructed where each point is associated with a different global state, and thus is an instance of the more restrictive structures used in [1].

formula φ is *satisfiable with respect to* $\mathcal{C}$ iff for some $\mathcal{I} \in \mathcal{C}$ and some point (r, n) in $\mathcal{I}$, we have $(\mathcal{I}, r, n) \models \varphi$.

We now turn our attention to formally defining the classes of interpreted systems of interest. For some of these definitions, it will be useful to give a number of equivalent presentations.

Perfect recall means, intuitively, that an agent's local state encodes everything that has happened (from that agent's point of view) thus far in the run. To make this precise, we need to define "what has happened so far from the agent's point of view." Let *agent i's local-state sequence at the point* (r, n) be the sequence $l_0, \ldots, l_k$ of states that agent i takes on in run r up to and including time n, with consecutive repetitions omitted. For example, if from times 0 through 4 in run r agent i goes through the sequence l, l, l', l, l of states, its history at $(r, 4)$ is just l, l', l. Agent i's local-state sequence at a point (r, m) essentially describes what has happened in the run up to time m, from i's point of view. Omitting consecutive repetitions from the local-state is intended to model asynchrony; "stuttering" is ignored.

Roughly speaking, agent i has perfect recall if i's current state encodes its history, i.e., i's whole local-state sequence. More formally, we say that *agent i has perfect recall* (alternatively, *agent i does not forget*) in system $\mathcal{R}$ if at all points (r, n) and (r', n') in $\mathcal{R}$, if $(r, n) \sim_i (r', n')$, then r has the same local-state sequence at both (r, n) and (r', n').

There are a number of equivalent characterizations of perfect recall. One characterization that will prove particularly useful in the comparison with the concept of no learning, which we are about to define, is the following. Let $S = (s_0, s_1, s_2, \ldots)$ and $T = (t_0, t_1, t_2, \ldots)$ be two (finite or infinite) sequences and let $\sim$ be a relation on the elements of S and T. Then we say that S and T are $\sim$-concordant if there is some k (k may be ∞) and nonempty consecutive intervals $S_1, \ldots, S_k$ of S and $T_1, \ldots, T_k$ of T such that for all $s \in S_j$ and $t \in T_j$, we have $s \sim t$ for $j = 1, \ldots, k$.

LEMMA 2.2 (see [5, 14]). *The following are equivalent.*

(a) *Agent i has perfect recall in system $\mathcal{R}$.*

(b) *For all points $(r, n) \sim_i (r', n')$ in $\mathcal{R}$, $((r, 0), \ldots, (r, n))$ is $\sim_i$-concordant with $((r', 0), \ldots, (r', n'))$.*

(c) *For all points $(r, n) \sim_i (r', n')$ in $\mathcal{R}$, if $n > 0$, then either $(r, n-1) \sim_i (r', n')$ or there exists a number $l < n'$ such that $(r, n-1) \sim_i (r', l)$ and for all k with $l < k \leq n'$ we have $(r, n) \sim_i (r', k)$.*

(d) *For all points $(r, n) \sim_i (r', n')$ in $\mathcal{R}$, if $k \leq n$, then there exists $k' \leq n'$ such that $(r, k) \sim_i (r', k')$.*

Proof. The implications from (a) to (b), from (b) to (c), and from (c) to (d) are straightforward. The implication from (d) to (a) can be proved by a straightforward induction on $n + n'$. □

Lemma 2.2 shows that perfect recall requires an unbounded number of local states in general, since agent i may have an infinite number of distinct histories in a given system. A system where agent i has perfect recall is shown in Figure 2.1, where the vertical lines denote runs (with time 0 at the top) and all points that i cannot distinguish are enclosed in the same region.

We remark that the official definition of perfect recall given here is taken from [1]. In [5], part (d) of Lemma 2.2 was taken as the definition of perfect recall (which was called "no forgetting" in that paper).

Roughly speaking, no learning is the dual notion to perfect recall. Perfect recall says that if the agent considers run r' possible at the point (r, n), in that there is a point (r', n') that the agent cannot distinguish from (r, n), then the agent must have

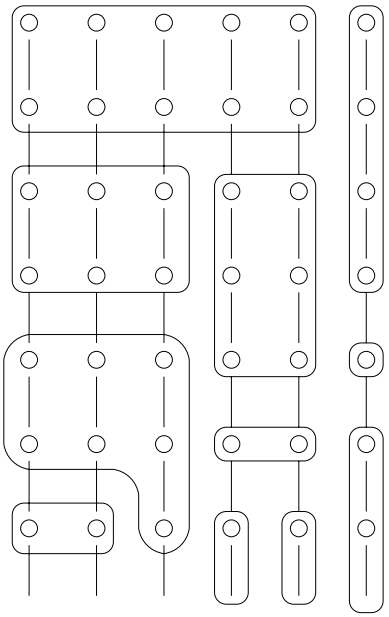


FIG. 2.1. *A system where agent i has perfect recall.*

considered r' possible at all times in the past (i.e., at all points (r, k) with $k \leq n$); it is not possible that the agent once considered r' impossible and then forgot this fact. No learning says that if the agent considers r' possible at (r, n), then the agent will consider r' possible at all times in the future; the agent will not learn anything that will allow him to distinguish r from r'. More formally, we define an agent's *future local-state sequence* at (r, n) to be the sequence of local states $l_0, l_1, \ldots$ that the agent takes on in run r, starting at (r, n), with consecutive repetitions omitted. We say agent i *does not learn* in system $\mathcal{R}$ if at all points (r, n) and (r', n') in $\mathcal{R}$, if $(r, n) \sim_i (r', n')$, then r has the same future local-state sequence at both (r, n) and (r', n').

Just as with perfect recall, there are a number of equivalent formulations of no learning.

LEMMA 2.3. *The following are equivalent.*

(a) *Agent i does not learn in system* $\mathcal{R}$.
(b) *For all points* $(r, n) \sim_i (r', n')$ *in* $\mathcal{R}$, $((r, n), (r, n+1), \ldots)$ *is* $\sim_i$*-concordant with* $((r', n'), (r', n'+1), \ldots)$.
(c) *For all points* $(r, n) \sim_i (r', n')$ *in* $\mathcal{R}$, *either* $(r, n+1) \sim_i (r', n')$ *or there exists a number* $l > n'$ *such that* $(r, n+1) \sim_i (r', l)$ *and for all* k *with* $l > k \geq n'$ *we have* $(r, n) \sim_i (r', k)$.

Notice that we have no analogue to part (d) of Lemma 2.2 in Lemma 2.3 (where $\leq$ is replaced by $\geq$). The analogue of (d) is strictly weaker than (a), (b), and (c), although they are equivalent in synchronous systems (which we are about to define formally). It was just this analogue of (d) that was used to define no learning in [5, 8]. We examine the differences between the notions carefully in section 6, where we provide more motivation for the definition chosen here.

In a *synchronous* system, we assume that every agent has access to a global clock that ticks at every instant of time, and the clock reading is part of its state. Thus,

in a synchronous system, each agent always "knows" the time. More formally, we say that a system $\mathcal{R}$ is *synchronous* if for all agents i and all points (r, n) and (r', n'), if $(r, n) \sim_i (r', n')$, then $n = n'$.[2] Observe that in a synchronous system where $(r, n) \sim_i (r', n)$, an easy induction on n shows that if i has perfect recall and $n > 0$, then $(r, n-1) \sim_i (r', n-1)$, while if i does not learn, then $(r, n+1) \sim_i (r', n+1)$.

Finally, we say that a system $\mathcal{R}$ has a *unique initial state* if for all runs $r, r' \in \mathcal{R}$, we have $r(0) = r'(0)$. Thus, if $\mathcal{R}$ is a system with a unique initial state, then we have $(r, 0) \sim_i (r', 0)$ for all runs r, r' in $\mathcal{R}$ and all agents i.

We say that $\mathcal{I} = (\mathcal{R}, \pi)$ is an interpreted system where agents have perfect recall (resp., agents do not learn, time is synchronous, there is a unique initial state) exactly if $\mathcal{R}$ is a system with that property. We use $\mathcal{C}_m$ to denote the class of all interpreted systems for m agents, and add the superscripts *nl*, *pr*, *sync*, and *uis* to denote particular subclasses of $\mathcal{C}_m$. Thus, for example, we use $\mathcal{C}_m^{nl,pr}$ to denote the set of all interpreted systems with m agents that have perfect recall and do not learn. We omit the subscript m when it is clear from context.

The results of [6, 7, 8] (some of which are based on earlier results of Ladner and Reif [11]) are summarized in Table 2.1. Each entry states a complexity class for which the corresponding problem is complete. For $\varphi \in KL_m$, we define $ad(\varphi)$ to be the greatest number of alternations of distinct K_i's along any branch in φ's parse tree. For example, $ad(K_1 \neg K_2 K_1 p) = 3$; temporal operators are not considered, so that $ad(K_1 \Box K_1 p) = 1$. (In Table 2.1, we do not consider the language CKL_1. This is because if $m = 1$, then $C\varphi$ is equivalent to $K_1\varphi$. Thus, CKL_1 is equivalent to KL_1.) We omit the definitions of complexity classes such as Π_1^1 and nonelementary time $ex(ad(\varphi) + 1, c|\varphi|)$ here; see [9] for details. (Note that c is a constant in the latter expression.) All that matters for our purposes is that for the cases where the complexity is Π_1^1 or co-r.e., there can be no recursive axiomatization; the validity problem is too hard. We provide complete axiomatizations here for the remaining cases.

TABLE 2.1
The complexity of the validity problem for logics of knowledge and time.

	CKL_m, $m \geq 2$	KL_m, $m \geq 2$	KL_1
$\mathcal{C}_m^{pr}$, $\mathcal{C}_m^{pr,sync}$, $\mathcal{C}_m^{pr,uis}$, $\mathcal{C}_m^{pr,sync,uis}$	Π_1^1	nonelementary time $ex(ad(\varphi)+1, c\|\varphi\|)$	double-exponential time
$\mathcal{C}_m^{nl}$, $\mathcal{C}_m^{nl,pr}$, $\mathcal{C}_m^{nl,pr,sync}$, $\mathcal{C}_m^{nl,sync}$	Π_1^1	nonelementary space $ex(ad(\varphi), c\|\varphi\|)$	EXPSPACE
$\mathcal{C}_m^{nl,pr,uis}$	Π_1^1	Π_1^1	EXPSPACE
$\mathcal{C}_m^{nl,uis}$	co-r.e.	co-r.e.	EXPSPACE
$\mathcal{C}_m^{nl,sync,uis}$, $\mathcal{C}_m^{nl,pr,sync,uis}$	EXPSPACE	EXPSPACE	EXPSPACE
$\mathcal{C}_m$, $\mathcal{C}_m^{sync}$, $\mathcal{C}_m^{sync,uis}$, $\mathcal{C}_m^{uis}$	EXPTIME	PSPACE	PSPACE

3. Axiom systems. In this section, we describe the axioms and inference rules that we need for reasoning about knowledge and time for various classes of systems,

[2] We remark that in [5], a slightly weaker definition is given: There, a system is said to be synchronous if for all runs r, if $(r, n) \sim_i (r, n')$, then $n = n'$. It is easy to show (by induction on n) that the two definitions are equivalent for systems where agents have perfect recall. In general, however, they are different. The definition given here is the one used in [1, 8].

and we state the completeness results. The proofs of these results are deferred to section 5.

For reasoning about knowledge alone, the following system, with axioms K1–K5 and rules of inference R1–R2, is well known to be sound and complete [1, 9]:

K1. All tautologies of propositional logic
K2. $K_i\varphi \wedge K_i(\varphi \Rightarrow \psi) \Rightarrow K_i\psi$, $i = 1, \ldots, m$
K3. $K_i\varphi \Rightarrow \varphi$, $i = 1, \ldots, n$
K4. $K_i\varphi \Rightarrow K_iK_i\varphi$, $i = 1, \ldots, m$
K5. $\neg K_i\varphi \Rightarrow K_i\neg K_i\varphi$, $i = 1, \ldots, m$
R1. From φ and $\varphi \Rightarrow \psi$ infer ψ
R2. From φ infer $K_i\varphi$, $i = 1, \ldots, m$

This axiom system is known as $\mathrm{S5}_m$.

For reasoning about the temporal operators individually, the following system (together with K1 and R1), is well known to be sound and complete [3]:

T1. $\bigcirc\varphi \wedge \bigcirc(\varphi \Rightarrow \psi) \Rightarrow \bigcirc\psi$
T2. $\bigcirc(\neg\varphi) \Leftrightarrow \neg\bigcirc\varphi$
T3. $\varphi\, U\, \psi \Leftrightarrow \psi \vee (\varphi \wedge \bigcirc(\varphi\, U\, \psi))$
RT1. From φ infer $\bigcirc\varphi$
RT2. From $\varphi' \Rightarrow \neg\psi \wedge \bigcirc\varphi'$ infer $\varphi' \Rightarrow \neg(\varphi\, U\, \psi)$

The system containing the above axioms and inference rules for both knowledge and time is called $\mathrm{S5}_m^U$. $\mathrm{S5}_m^U$ is easily seen to be sound for $\mathcal{C}_m$, the class of all systems for m agents. Given that there is no necessary connection between knowledge and time in $\mathcal{C}_m$, it is perhaps not surprising that $\mathrm{S5}_m^U$ should be complete with respect to $\mathcal{C}_m$ as well. Interestingly, even if we impose the requirements of synchrony or uis, $\mathrm{S5}_m^U$ remains complete; our language is not rich enough to capture these conditions.

THEOREM 3.1. *$\mathrm{S5}_m^U$ is a sound and complete axiomatization for the language KL_m with respect to $\mathcal{C}_m$, $\mathcal{C}_m^{sync}$, $\mathcal{C}_m^{uis}$, and $\mathcal{C}_m^{sync,uis}$.*

We get the same lack of interaction between knowledge in the classes $\mathcal{C}_m$, $\mathcal{C}_m^{sync}$, $\mathcal{C}_m^{uis}$, and $\mathcal{C}_m^{sync,uis}$ even when we add common knowledge. It is well known that the following two axioms and inference rule characterize common knowledge [1, 4]:

C1. $E\varphi \Leftrightarrow \bigwedge_{i=1}^{m} K_i\varphi$
C2. $C\varphi \Rightarrow E(\varphi \wedge C\varphi)$
RC1. From $\varphi \Rightarrow E(\psi \wedge \varphi)$ infer $\varphi \Rightarrow C\psi$

Let $\mathrm{S5C}_m^U$ be the result of adding C1, C2, and RC1 to $\mathrm{S5}_m^U$. We then have the following extension of Theorem 3.1.

THEOREM 3.2. *$\mathrm{S5C}_m^U$ is a sound and complete axiomatization for the language CKL_m with respect to $\mathcal{C}_m$, $\mathcal{C}_m^{sync}$, $\mathcal{C}_m^{uis}$, and $\mathcal{C}_m^{sync,uis}$.*

If we restrict our attention to systems with perfect recall or no learning, then knowledge and time do interact. We start by stating five axioms of interest and then discuss them.

KT1. $K_i\Box\varphi \Rightarrow \Box K_i\varphi$, $i = 1, \ldots, m$
KT2. $K_i\bigcirc\varphi \Rightarrow \bigcirc K_i\varphi$, $i = 1, \ldots, m$
KT3. $K_i\varphi_1 \wedge \bigcirc(K_i\varphi_2 \wedge \neg K_i\varphi_3) \Rightarrow L_i((K_i\varphi_1)\, U\, [(K_i\varphi_2)\, U\, \neg\varphi_3])$, $i = 1, \ldots, m$
KT4. $K_i\varphi_1\, U\, K_i\varphi_2 \Rightarrow K_i(K_i\varphi_1\, U\, K_i\varphi_2)$, $i = 1, \ldots, m$
KT5. $\bigcirc K_i\varphi \Rightarrow K_i\bigcirc\varphi$, $i = 1, \ldots, m$

Axiom KT1 was first discussed by Ladner and Reif [11]. Informally, this axiom states that if a proposition is known to be always true, then it is always known to be true. It is not hard to show, using Lemma 2.2, that axiom KT1 holds with perfect recall, that is, KT1 is valid in $\mathcal{C}_m^{pr}$. It was conjectured in an early draft of [1] that the

system $S5^U_m + KT1$ would be complete for $\mathcal{C}^{pr}_m$. However, it was shown in [14] that this conjecture was false. To get completeness we need a stronger axiom: KT3.

It is not hard to see that KT3 is valid in systems with perfect recall. A formal proof is provided in section 5, but we can give some intuition here. Suppose that $(\mathcal{I}, r, n) \models K_i\varphi_1 \wedge \bigcirc(K_i\varphi_2 \wedge \neg K_i\varphi_3)$. That means that $(\mathcal{I}, r, n+1) \models \neg K_i\varphi_3$, so there must be some point $(r', n') \sim_i (r, n+1)$ such that $(\mathcal{I}, r', n') \models \neg\varphi_3$. Because agent i has perfect recall, there must exist some $k' \leq n'$ such that $(r', k') \sim_i (r, n)$. It is not hard to show, using Lemma 2.2(c), that $(\mathcal{I}, r', k') \models K_i\varphi_1\, U\, (K_i\varphi_2\, U\, \neg\varphi_3)$. It follows that $(\mathcal{I}, r, n) \models L_i(K_i\varphi_1\, U\, (K_i\varphi_2\, U\, \neg\varphi_3))$.

In the presence of the other axioms, KT3 implies KT1. The following general lemma, which applies to all the proof systems we consider, is useful for the proof.

LEMMA 3.3. *Suppose that* $\vdash \varphi_1 \Leftrightarrow \varphi_2$ *and let* ψ' *be the result of replacing some of the occurrences of* φ_1 *in* ψ *by* φ_2. *Then* $\vdash \psi \Leftrightarrow \psi'$.

Proof. By induction, it suffices to assume that ψ' is the result of replacing one occurrence of φ_1 in ψ by φ_2. The proof proceeds by a straightforward induction on the structure of ψ. □

LEMMA 3.4. KT1 *is provable in* $S5^U_m + KT3$.

Proof. Note that by purely temporal reasoning, we can show $\vdash \Box\varphi \Leftrightarrow \Box\Box\varphi$. Using R2 and K2, this implies that $\vdash K_i\Box\varphi \Leftrightarrow K_i\Box\Box\varphi$. Now if $\varphi_1 = \varphi_2 = \mathit{true}$, then (using 3.3) KT3 simplifies to $\bigcirc\neg K_i\varphi_3 \Rightarrow \neg K_i\Box\varphi_3$. In particular, taking the contrapositive, substituting $\varphi_3 = \Box\varphi$, and using T2, we obtain $\vdash K_i\Box\Box\varphi \Rightarrow \bigcirc K_i\Box\varphi$, which yields $\vdash K_i\Box\varphi \Rightarrow \bigcirc K_i\Box\varphi$ by the equivalence noted above. It is also straightforward to show that $\vdash \Box\varphi \Rightarrow \varphi$, from which it follows, using K2 and R2, that $\vdash K_i\Box\varphi \Rightarrow K_i\varphi$. The axiom KT1 now follows using the rule RT2 (using the fact that $\Box\varphi$ is an abbreviation for $\neg(\mathit{true}\, U\, \neg\varphi)$). □

KT3 turns out to be strong enough to give us completeness, with or without the condition uis.

THEOREM 3.5. $S5^U_m + KT3$ *is a sound and complete axiomatization for the language* KL_m *with respect to* $\mathcal{C}^{pr}_m$ *and* $\mathcal{C}^{pr,uis}_m$.

Theorem 3.1 shows that requiring synchrony or uis does not have an impact when we consider the class of all systems—$\mathcal{C}_m$, $\mathcal{C}^{sync}_m$, $\mathcal{C}^{uis}_m$, and $\mathcal{C}^{sync,uis}_m$ are all axiomatized by $S5^U_m$—and Theorem 3.5 shows that adding uis does not have an impact in the presence of perfect recall. However, requiring synchrony does have an impact in the presence of perfect recall. It is easy to see that KT2 is valid in $\mathcal{C}^{pr,sync}_m$, and it clearly is not valid in $\mathcal{C}^{pr}_m$. Moreover, KT2 suffices for completeness in $\mathcal{C}^{pr,sync}_m$; we do not need the complications of KT3.

THEOREM 3.6. $S5^U_m + KT2$ *is a sound and complete axiomatization for the language* KL_m *with respect to* $\mathcal{C}^{pr,sync}_m$ *and* $\mathcal{C}^{pr,sync,uis}_m$.

KT4 is the axiom that characterizes no learning. More precisely, we have the following.

THEOREM 3.7. $S5^U_m + KT4$ *is a sound and complete axiomatization for the language* KL_m *with respect to* $\mathcal{C}^{nl}_m$.

Unlike previous cases, the uis assumption is not innocuous in the presence of nl. For one thing, it is not hard to check that assuming uis leads to extra properties. Indeed, as Table 2.1 shows, if $m \geq 2$, then assuming a unique initial state along with no learning results in a class of systems that do not have a recursive axiomatic characterization, since the validity problem is co-r.e.-complete. On the other hand, if there is only one agent in the picture, things simplify. No learning together with uis implies perfect recall. Thus, we get the following.

THEOREM 3.8. $S5_m^U + KT3 + KT4$ *is a sound and complete axiomatization for the language* KL_m *with respect to* $\mathcal{C}_m^{nl,pr}$. *Moreover, it is a sound and complete axiomatization for the language* KL_1 *with respect to* $\mathcal{C}_1^{nl,pr,uis}$.

In synchronous systems with no learning, things again become simpler. KT5, the converse of KT2, suffices to characterize such systems.

THEOREM 3.9. $S5_m^U + KT5$ *is a sound and complete axiomatization for the language* KL_m *with respect to* $\mathcal{C}_m^{nl,sync}$.

Of course, it follows from Theorem 3.9 that KT4 can be derived in the system $S5_m^U + KT5$ (although this result takes some work to prove directly).

Not surprisingly, if we combine perfect recall, no learning, and synchrony, then KT2 and KT5 give us a complete axiomatization.

THEOREM 3.10. $S5_m^U + KT2 + KT5$ *is a sound and complete axiomatization for the language* KL_m *with respect to* $\mathcal{C}_m^{nl,pr,sync}$.

Finally, it can be shown that when we combine no learning, synchrony, and uis, then not only do all agents consider the same worlds possible initially, but they consider the same worlds possible at all times. As a result, the axiom $K_i\varphi \Leftrightarrow K_j\varphi$ is valid in this case. This allows us to reduce to the single-agent case. Moreover, as we observed above, in the single-agent case, no learning and uis imply perfect recall. Thus, we get the following result.

THEOREM 3.11. $S5_m^U + KT2 + KT5 + \{K_i\varphi \Leftrightarrow K_1\varphi\}$ *is a sound and complete axiomatization for the language* KL_m *with respect to* $\mathcal{C}_m^{nl,sync,uis}$ *and* $\mathcal{C}_m^{nl,pr,sync,uis}$.

A glance at Table 2.1 shows that we have now provided axiomatizations for all the cases where complete axiomatizations exist. (Notice that for the language CKL_m, if $m = 1$, then common knowledge reduces to knowledge, while if $m > 1$, then complete axiomatizations can exist only for $\mathcal{C}_m$, $\mathcal{C}_m^{sync}$, $\mathcal{C}_m^{uis}$, $\mathcal{C}_m^{sync,uis}$, $\mathcal{C}_m^{nl,sync,uis}$, and $\mathcal{C}_m^{nl,pr,sync,uis}$. The first four cases were dealt with in Theorem 3.2, while in the last two, as we have observed, common knowledge reduces to the knowledge of agent 1.)

4. A framework for completeness proofs. In this section we develop a general framework for completeness proofs that reduces the work required in each of the different completeness results to a single lemma. With respect to the temporal dimension, our constructions resemble those previously used for completeness of dynamic logic [10] and temporal logics, in that we construct a model for a consistent formula ψ out of consistent subsets of a finite set of formulas, called the *closure* of ψ. However, in order to deal with the knowledge modalities, we need a number of distinct levels of closure, having a tree-like structure. At the leaves of this tree-like structure, the closure is like the usual closure for temporal logic. As we move towards the root, we add formulas to the closure that increase the level of nesting of the knowledge modalities.

A formula ψ is said to be *consistent* in a logic $\mathcal{L}$ if it is not the case that $\vdash_{\mathcal{L}} \neg\psi$. For each of the pairs $(\mathcal{L}, \mathcal{C})$ of logic $\mathcal{L}$ and class $\mathcal{C}$ of systems we consider, the proof that $\mathcal{L}$ is complete with respect to $\mathcal{C}$ proceeds by constructing, for every formula ψ consistent with respect to $\mathcal{L}$, a system in $\mathcal{C}$ containing a point at which ψ is true. All the results in this section hold for every logic containing $S5_m^U$, except for Lemma 4.8, which mentions common knowledge. This lemma holds for every logic containing $S5C_m^U$. Rather than mentioning the logic $\mathcal{L}$ explicitly in each case, we just write $\vdash$ rather than $\vdash_{\mathcal{L}}$; the intended logic(s) will be clear from context. We also fix the formula ψ, which is assumed to be consistent with respect to $\mathcal{L}$.

A finite sequence $\sigma = i_1 i_2 \ldots i_k$ of agents, possibly equal to the null sequence ϵ, is called an *index* if $i_l \neq i_{l+1}$ for all $l < k$. We write $|\sigma|$ for the length k of such a sequence; the null sequence has length equal to 0.

If S is a set, and S^* is the set of all finite sequences over S, we define the absorptive concatenation function $\#$ from $S^* \times S$ to S^* as follows. Given a sequence σ in S^* and an element x of S, we take $\sigma\#x = \sigma$ if the final element of σ is x. If the final element of σ is not equal to x, then we take $\sigma\#x$ to be σx, i.e., the result of concatenating x to σ. We write simply $x_1\#x_2\#x_3\#\ldots\#x_n$ for $(\ldots((x_1\#x_2)\#x_3)\ldots)\#x_n$. We shall have two distinct uses for this function, applying it primarily to sequences of agents, and sometimes to sequences of "instantaneous states" of agents in the context of asynchronous systems.

If $\psi \in CKL_m$, for each $k \geq 0$, we define the k-closure $cl_k(\psi)$, and for each agent i, we define the k,i-closure $cl_{k,i}(\psi)$. The definition of these sets proceeds by mutual recursion: First, we let the *basic closure* $cl_0(\psi)$ be the smallest set containing ψ that is closed under subformulas, contains $\neg\varphi$ if it contains φ and φ is not of the form $\neg\varphi'$, contains $EC\varphi$ if it contains $C\varphi$, and contains $K_1\varphi, \ldots, K_m\varphi$ if it contains $E\varphi$. (Of course, the last two clauses do not apply if ψ is in KL_m, and thus does not mention common knowledge.) If i is an agent, we take $cl_{k,i}(\psi)$ to be the union of $cl_k(\psi)$ with the set of formulas of the form $K_i(\varphi_1 \vee \cdots \vee \varphi_n)$ or $\neg K_i(\varphi_1 \vee \cdots \vee \varphi_n)$, where the φ_l are distinct formulas in $cl_k(\psi)$. (It is not necessary to close under subformulas here, since the disjunctions in these formulas are already "decided" in the sense defined below.) Finally, $cl_{k+1}(\psi)$ is defined to be $\bigcup_{i=1}^{m} cl_{k,i}(\psi)$.

If X is a finite set of formulas, we write φ_X for the conjunction of the formulas in X. A finite set X of formulas is said to be consistent if φ_X is consistent. If X is a finite set of formulas and φ is a formula, we write $X \Vdash \varphi$ when $\vdash \varphi_X \Rightarrow \varphi$.[3] A finite set Cl of formulas is said to be *negation closed* if, for all $\varphi \in Cl$, either $\neg\varphi \in Cl$ or φ is of the form $\neg\varphi'$ and $\varphi' \in Cl$. (Note that the sets $cl_k(\psi)$ and $cl_{k,i}(\psi)$ are negation closed.) We define an *atom* of Cl to be a maximal consistent subset of Cl.

The following lemma collects a number of obvious facts that we typically use without comment.

LEMMA 4.1. *Suppose that X is a finite set of formulas and Cl is a negation-closed set of formulas.*

(a) *If $X \Vdash \varphi_1$ and $\vdash \varphi_1 \Rightarrow \varphi_2$, then $X \Vdash \varphi_2$.*

(b) *If X is an atom of Cl and $\varphi \in Cl$, then either $X \Vdash \varphi$ or $X \Vdash \neg\varphi$.*

(c) *If $\vdash$ denotes provability in a proof system containing* K1 *and* R1*, then $\vdash \bigvee_{X \text{ an atom of } Cl} \varphi_X$.*

Proof. All parts of the lemma are quite easy. We remark that (c) follows from the observation that $\bigvee_{X \text{ an atom of } Cl} \varphi_X$ is equivalent to *true*, which can be easily proved using only propositional reasoning (K1 and R1). □

We begin the construction of the model of ψ by first constructing a *premodel*, which is a structure $\langle S, \rightarrow, \approx_1, \ldots, \approx_n\rangle$ consisting of a set S of states, a binary relation $\rightarrow$ on S, and for each agent i an equivalence relation $\approx_i$ on S. Recall from section 2 that for a formula $\varphi \in KL_m$, the alternation depth $ad(\varphi)$ is the number of alternations of distinct operators K_i in φ. Let $d = ad(\psi)$ if $\psi \in KL_m$; otherwise (that is, if ψ mentions the modal operator C), let $d = 0$.

The set S consists of all the pairs (σ, X) such that σ is an index, $|\sigma| \leq d$, and

1. if $\sigma = \epsilon$, then X is an atom of $cl_d(\psi)$; and
2. if $\sigma = \tau i$, then X is an atom of $cl_{k,i}(\psi)$, where $k = d - |\sigma|$.

We can partition the set S of states into sets S_σ according to the first component;

[3] Note that $X \Vdash \varphi$ is not equivalent to $X \vdash \varphi$ (under perhaps the most natural definition of $\vdash$ with sets of formulas on the left-hand side) because of generalization rules like R2 and RT1. For example, although $\varphi \vdash K_i\varphi$, it is not the case that $\vdash \varphi \Rightarrow K_i\varphi$.

that is, $S_\sigma = \{(\sigma, X) \mid (\sigma, X) \in S\}$. The indices σ put a tree-like structure on this collection. Note that as $|\sigma|$ decreases, the size of the closure from which the atoms X are drawn increases.

The relation $\rightarrow$ is defined so that $(\sigma, X) \rightarrow (\tau, Y)$ iff $\tau = \sigma$ and the formula $\varphi_X \wedge \bigcirc\varphi_Y$ is consistent. If X is an atom, we write X/K_i for the set of formulas φ such that $K_i\varphi \in X$. We say that states (σ, X) and (τ, Y) are *i-adjacent* if $\sigma\#i = \tau\#i$. The relation $\approx_i$ is defined so that $(\sigma, X) \approx_i (\tau, Y)$ iff σ and τ are i-adjacent and $X/K_i = Y/K_i$. Clearly, i-adjacency is an equivalence relation, as is the relation $\approx_i$.

A *σ-state (for ψ)* is a pair (σ, X) as above. A state (for ψ) is a σ-state for some index σ with $|\sigma| \leq d$. Thus (σ, X) is the unique σ-state with atom X. If $s = (\sigma, X)$ is a state, we define φ_s to be the formula φ_X, and write $s \Vdash \varphi$ for $\vdash \varphi_s \Rightarrow \varphi$. We say that the state s *directly decides* a formula φ if either (a) $\varphi \in X$, (b) $\neg\varphi \in X$, or (c) $\varphi = \neg\varphi'$ and $\varphi' \in X$. We say that s *decides* φ if either $s \Vdash \varphi$ or $s \Vdash \neg\varphi$. Clearly, if s directly decides φ, then s decides φ. Note that if $\sigma = \tau i$, then, by Lemma 4.1(b), each σ-state directly decides every formula in $cl_{d-|\sigma|,i}(\psi)$. Also, every ϵ-state directly decides every formula in $cl_d(\sigma)$.

LEMMA 4.2. *If s and t are i-adjacent states, then the same formulas of the form $K_i\varphi$ are directly decided by s and t.*

Proof. Suppose that s and t are i-adjacent, $s = (\sigma, X)$ and $t = (\tau, Y)$. By definition, we have that either (i) $\sigma = \tau$, (ii) $\sigma = \tau i$, or (iii) $\sigma i = \tau$. Clearly if $\sigma = \tau$, then s and t directly decide the same formulas (and, a fortiori, the same formulas of the form $K_i\varphi$) since they are both maximal consistent subsets of the same set of formulas. If $\sigma \neq \tau$, then either $\sigma = \tau i$ or $\tau = \sigma i$. By symmetry, it suffices to deal with the case $\sigma = \tau i$. By definition, s directly decides the K_i-formulas in $cl_{d-|\sigma|,i}(\psi)$, while t directly decides the K_i-formulas in $cl_{d-|\tau|,j}(\psi)$ if $\tau = \tau' j$ or $cl_d(\psi)$ if $\tau = \epsilon$. Thus, it suffices to show that, for all formulas φ and agents j, we have that $K_i\varphi \in cl_{k,i}(\psi)$ iff $K_i\varphi \in cl_{k+1,j}$ if $k < d-1$ and that $K_i\varphi \in cl_{d-1,i}(\psi)$ iff $K_i\varphi \in cl_d(\psi)$. This is immediate from the definitions. □

If s is a σ-state, we take $\Phi_{s,i}$ to be the disjunction of the formulas φ_t, where t ranges over the σ-states satisfying $s \approx_i t$, and we take $\Phi^+_{s,i}$ to be the disjunction of the formulas φ_t, where t ranges over the $(\sigma\#i)$-states satisfying $s \approx_i t$.[4] Observe that because $\approx_i$ is an equivalence relation we have that if $s \approx_i t$, then $\Phi_{s,i} = \Phi_{t,i}$ and $\Phi^+_{s,i} = \Phi^+_{t,i}$. The following result lists a number of knowledge formulas decided by states.

LEMMA 4.3.

(a) *If s is a σ-state and t is a σ-state or $(\sigma\#i)$-state such that $s \not\approx_i t$, then $s \Vdash K_i\neg\varphi_t$.*
(b) *For all σ-states s, we have $s \Vdash K_i\Phi_{s,i}$; in addition, if $|\sigma\#i| \leq d$, then $s \Vdash K_i\Phi^+_{s,i}$.*
(c) *For all σ-states s and $(\sigma\#i)$-states t with $s \approx_i t$, we have $s \Vdash L_i\varphi_t$.*
(d) *If s is a σ-state and t is a $(\sigma\#i)$-state such that $s \not\approx_i t$, then $t \Vdash \neg K_i\Phi^+_{s,i}$.*

Proof. For (a), suppose that $s \not\approx_i t$, where $s = (\sigma, X)$ and $t = (\tau, Y)$, where τ is either σ or $\sigma\#i$. Then $X/K_i \neq Y/K_i$ so either there exists a formula $K_i\varphi \in X$ such that $K_i\varphi \notin Y$ or there exists a formula $K_i\varphi \in Y$ such that $K_i\varphi \notin X$. As the states s and t are i-adjacent, by Lemma 4.2, in either case the formula $K_i\varphi$ is directly decided by both the states s and t. In the first case, since $K_i\varphi \notin Y$ and $K_i\varphi$ is directly decided by t, it follows that $\neg K_i\varphi \in Y$, and hence that $\vdash \varphi_t \Rightarrow \neg K_i\varphi$. Using R2, it

[4] It can be shown that if $|\sigma\#i| \leq d$, then $\Phi_{s,i}$ is logically equivalent to $\Phi^+_{s,i}$, but we do not need this fact here.

follows that $\vdash K_i(K_i\varphi \Rightarrow \neg\varphi_t)$. By K4 we obtain from the fact that $K_i\varphi \in X$ that $s \Vdash K_iK_i\varphi$. It now follows using K2 that $s \Vdash K_i\neg\varphi_t$. In the second case, we have that $\vdash \varphi_t \Rightarrow K_i\varphi$, and hence, using R2, that $\vdash K_i(\neg K_i\varphi \Rightarrow \neg\varphi_t)$. By K4 we obtain from the fact that $\neg K_i\varphi \in X$ that $s \Vdash K_i\neg K_i\varphi$. It now follows using K2 that $s \Vdash K_i\neg\varphi_t$.

For (b), by Lemma 4.1(c), we have that $\vdash \bigvee_{X \text{ an atom of } cl_{k,i}(\psi)} \varphi_X$. Hence, by R2 we obtain that $\vdash K_i \bigvee_{\sigma\text{-states } t} \varphi_t$. It follows from this using (a) and K2 that $s \Vdash K_i\Phi_{s,i}$. If $|\sigma\#i| \leq d$, then a similar argument shows that $s \Vdash K_i\Phi^+_{s,i}$.

For (c), suppose that $s = (\sigma, X) \approx_i (\sigma\#i, Y) = t$ and $k = d - |\sigma\#i|$. We claim first that if $W = Y \cap cl_k(\psi)$, then $s \Vdash K_i\neg\varphi_t \Leftrightarrow K_i\neg\varphi_W$. This is because the fact that Y is a subset of $cl_{k,i}(\psi)$ implies that all formulas φ in $Y \setminus W$ are of the form $K_i\varphi'$ or $\neg K_i\varphi'$, and hence $\varphi \in X$ iff $\varphi \in Y$. Also, by K4 and K5 we have that $s \Vdash K_iK_i\varphi'$ when $K_i\varphi' \in X$ and $s \Vdash K_i\neg K_i\varphi'$ when $K_i\varphi' \notin X$. It follows using K2 that $s \Vdash K_i\varphi_{Y\setminus W}$. Since φ_t is equivalent to $\varphi_W \wedge \varphi_{Y\setminus W}$, we obtain using K2 that $s \Vdash K_i\neg\varphi_t \Leftrightarrow K_i\neg\varphi_W$.

Now by K3 we have $\vdash \varphi_t \Rightarrow L_i\varphi_t$. Further, the argument of the previous paragraph also shows $t \Vdash K_i\neg\varphi_t \Leftrightarrow K_i\neg\varphi_W$, so we obtain that $t \Vdash L_i\varphi_W$. But $\neg\varphi_W$ is equivalent to the disjunction of a subset $\{\varphi_1, \ldots, \varphi_n\}$ of $cl_k(\psi)$. Let α be the formula $K_i(\varphi_1 \vee \cdots \vee \varphi_n)$, which is equivalent to $K_i\neg\varphi_W$. It follows from the definition of $cl_{k,i}(\psi)$ that α is in $cl_{k,i}(\psi)$, and hence directly decided by both t and s. Consequently, α is not in Y, since $t \Vdash \neg\alpha$. Because $X/K_i = Y/K_i$, the formula α is not in X either, so $s \Vdash \neg\alpha$. Applying the fact that α is equivalent to $K_i\neg\varphi_W$, we see that $s \Vdash L_i\varphi_W$. The equivalence of the previous paragraph now yields that $s \Vdash L_i\varphi_t$.

For (d), note that if t and v are distinct $(\sigma\#i)$-states, then $t \Vdash \neg\varphi_v$. Thus, if s is a σ-state such that $s \not\approx_i t$, then $t \Vdash \neg\Phi^+_{s,i}$, which implies, using K3, that $t \Vdash \neg K_i\Phi^+_{s,i}$. □

If T is a set of states, then we write φ_T for the disjunction of the formulas φ_t for t in T. Using RT1, T1, and T2, the following result is immediate from the fact that $s \nrightarrow t$ implies $\vdash \varphi_s \Rightarrow \neg\bigcirc\varphi_t$, together with the fact that $\vdash \bigvee_{s \text{ a } \sigma\text{-state}} \varphi_s$, which follows from Lemma 4.1(c).

LEMMA 4.4. *Let s be a state and let T be the set of states t such that $s \to t$. Then $s \Vdash \bigcirc\varphi_T$.*

The next result provides a useful way to derive formulas containing the "until" operator.

LEMMA 4.5. *For all formulas α, β, and γ, if $\vdash \alpha \Rightarrow (\neg\gamma \wedge \bigcirc(\alpha \vee (\neg\beta \wedge \neg\gamma)))$, then $\vdash \alpha \Rightarrow \neg(\beta\, U\, \gamma)$.*

Proof. Suppose that $\vdash \alpha \Rightarrow \neg\gamma \wedge \bigcirc(\alpha \vee (\neg\beta \wedge \neg\gamma))$. By T3, we obtain that $\vdash \alpha \wedge (\beta\, U\, \gamma) \Rightarrow \neg\gamma \wedge \bigcirc(\beta\, U\, \gamma) \wedge \bigcirc(\alpha \vee (\neg\beta \wedge \neg\gamma))$. Since, by T3 again, $\vdash \beta\, U\, \gamma \Rightarrow \neg(\neg\beta \wedge \neg\gamma)$, it follows using T1 and RT1 that $\vdash \alpha \wedge (\beta U \gamma) \Rightarrow \neg\gamma \wedge \bigcirc(\alpha \wedge (\beta U \gamma))$. Now using RT2 we obtain $\vdash \alpha \wedge (\beta U \gamma) \Rightarrow \neg(\beta U \gamma)$, which implies that $\vdash \alpha \Rightarrow \neg(\beta U \gamma)$. □

The following shows that the premodel has properties resembling those for the truth definitions for formulas in the basic closure. Note that every state directly decides all formulas in the basic closure. Define a *$\to$-sequence* of states to be a (finite or infinite) sequence $s_1, s_2, \ldots$ such that $s_1 \to s_2 \to \cdots$.

LEMMA 4.6. *For all σ-states s, we have the following:*

(a) *If $\bigcirc\varphi \in cl_0(\psi)$, then for all states t such that $s \to t$, we have $s \Vdash \bigcirc\varphi$ iff $t \Vdash \varphi$.*

(b) *If $K_i\varphi \in cl_0(\psi)$, then $s \Vdash \neg K_i\varphi$ iff there is some σ-state t such that $s \approx_i t$ and $t \Vdash \neg\varphi$. Moreover, if $|\sigma\#i| \leq d$, then $s \Vdash \neg K_i\varphi$ iff there is some $(\sigma\#i)$-state t such that $s \approx_i t$ and $t \Vdash \neg\varphi$.*

(c) *If $\varphi_1 U \varphi_2 \in cl_0(\psi)$, then $s \Vdash \varphi_1 U \varphi_2$ iff there exists a $\rightarrow$-sequence $s = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$, where $n \geq 0$, such that $s_n \Vdash \varphi_2$, and $s_k \Vdash \varphi_1$ for all $k < n$.*

Proof. For part (a), suppose first that $s \Vdash \bigcirc\varphi$ and $s \rightarrow t$. Since $\varphi \in cl_0(\psi)$, it follows that $t \Vdash \varphi$ or $t \Vdash \neg\varphi$. However, by T1 and T2, the latter would contradict the assumption that $\varphi_s \wedge \bigcirc\varphi_t$ is consistent. Hence we have $t \Vdash \varphi$. Conversely, suppose that $s \rightarrow t$ and $t \Vdash \varphi$. Using T1, we have $\vdash \bigcirc\varphi_t \Rightarrow \bigcirc\varphi$. Since $\bigcirc\varphi \in cl_0(\psi)$ we have either $s \Vdash \bigcirc\varphi$ or $s \Vdash \neg\bigcirc\varphi$. But the latter would contradict $s \rightarrow t$, so we obtain $s \Vdash \bigcirc\varphi$.

For the "if" direction of part (b), note that the fact that $K_i\varphi$ is in $cl_0(\psi)$ implies that if $s \approx_i t$ and $s \Vdash K_i\varphi$, then $t \Vdash K_i\varphi$, and hence $t \Vdash \varphi$ by K3. For the converse, suppose that $t \Vdash \varphi$ for all σ-states t with $s \approx_i t$. Then $\vdash \Phi_{s,i} \Rightarrow \varphi$, and hence $\vdash K_i\Phi_{s,i} \Rightarrow K_i\varphi$, using K2 and R2. By Lemma 4.3(b), we have $s \Vdash K_i\Phi_{s,i}$. It follows immediately that $s \Vdash K_i\varphi$. If $|\sigma\#i| \leq d$, a similar argument shows that if $t \Vdash \varphi$ for all $(\sigma\#i)$-states t such that $s \approx_i t$, then $s \Vdash K_i\varphi$.

For part (c), note that if $\varphi_1 U \varphi_2$ is in $cl_0(\psi)$, then every state directly decides each of the formulas φ_1, φ_2, and $\varphi_1 U \varphi_2$. We first show that if there exists a sequence of states $s = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$ such that $s_n \Vdash \varphi_2$ and $s_k \Vdash \varphi_1$ for all $k < n$, then $s \Vdash \varphi_1 U \varphi_2$. We proceed by induction on n. The case $n = 0$ is immediate from T3. For the general case, notice that it follows from the induction hypothesis that $s_1 \Vdash (\varphi_1 U \varphi_2)$. Since $s_0 \rightarrow s_1$, it follows that $\varphi_{s_0} \wedge \bigcirc(\varphi_1 U \varphi_2)$ is consistent. By assumption, we also have $s_0 \Vdash \varphi_1$. Using T3, we see that $s_0 \Vdash \neg(\varphi_1 U \varphi_2)$ would be a contradiction. Hence $s_0 \Vdash \varphi_1 U \varphi_2$.

The converse follows immediately from Lemma 4.7 below. □

LEMMA 4.7. *If $\varphi_s \wedge (\varphi_1 U \varphi_2)$ is consistent, then there exists a $\rightarrow$-sequence $s = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$, such that $\varphi_{s_n} \wedge \varphi_2$ is consistent, and $\varphi_{s_k} \wedge \varphi_1$ is consistent for all $k < n$.*

Proof. Suppose by way of contradiction that $\varphi_s \wedge (\varphi_1 U \varphi_2)$ is consistent and no appropriate $\rightarrow$-sequence exists. Let T be the smallest set S of states such that (i) $s \in S$, and (ii) if $t \in S$, $t \rightarrow u$, and $s_u \wedge \varphi_1$ is consistent, then $u \in S$. Then we have that $t \Vdash \neg\varphi_2$ for all $t \in T$, so $\vdash \varphi_T \Rightarrow \neg\varphi_2$. In addition, for each $t \in T$ and state u such that $t \rightarrow u$, we have either $u \in T$ or $u \Vdash \neg\varphi_1 \wedge \neg\varphi_2$. Thus, using Lemma 4.4, we obtain $\vdash \varphi_T \Rightarrow \bigcirc(\varphi_T \vee (\neg\varphi_1 \wedge \neg\varphi_2))$. It now follows using Lemma 4.5 that $\vdash \varphi_T \Rightarrow \neg(\varphi_1 U \varphi_2)$. In particular, since $s \in T$, we have $s \Vdash \neg(\varphi_1 U \varphi_2)$, which contradicts the assumption that $\varphi_s \wedge (\varphi_1 U \varphi_2)$ is consistent. □

For the next result, recall that when the formula ψ contains the common knowledge operator we take $d = 0$, so that all states are ϵ-states.

LEMMA 4.8. *If $C\varphi \in cl_0(\psi)$, then $s \Vdash \neg C\varphi$ iff there is a state t reachable from s through the relations $\approx_i$ such that $t \Vdash \neg\varphi$.*

Proof. The implication from right to left is a straightforward consequence of the fact that if $t \Vdash \neg\varphi$, then $t \Vdash \neg C\varphi$, by C1, C2, and K3, together with the fact that if $t \approx_i t'$, then $t \Vdash C\varphi$ iff $t' \Vdash C\varphi$. (Proof of the latter fact: If $t \Vdash C\varphi$, then $t \Vdash K_iC\varphi$ by C1 and C2. Hence, since $t \approx_i t'$ and $K_iC\varphi \in cl_0(\psi)$, we must have $t' \Vdash C\varphi$. The opposite direction follows symmetrically.) This leaves only the implication from left to right, for which we prove the contrapositive. Suppose that no state containing $\neg\varphi$ is reachable from s by means of a sequence of steps through the relations $\approx_i$. Let T be the set of states reachable from s. By Lemma 4.3(a), if t and t' are states with $t \not\approx_i t'$, then $t \Vdash K_i\neg\varphi_{t'}$. It follows from this that $t \Vdash K_i\varphi_T$ for every state $t \in T$ and agent i. Thus, because $\vdash \varphi_T \Rightarrow \varphi$ we have $\vdash \varphi_T \Rightarrow E(\varphi_T \wedge \varphi)$. By RC1 it follows that $\vdash \varphi_T \Rightarrow C\varphi$. Since $s \in T$, it is immediate that $s \Vdash C\varphi$. □

We say that an infinite $\rightarrow$-sequence of states $(s_0, s_1, \ldots)$, where $s_n = (\sigma, X_n)$ for all n, is *acceptable* if for all $n \geq 0$, if $\varphi_1 \, U \, \varphi_2 \in X_n$, then there exists an $m \geq n$ such that $s_m \Vdash \varphi_2$ and $s_k \Vdash \varphi_1$ for all k with $n \leq k < m$.

DEFINITION 4.9. *An* enriched system *for ψ is a pair $(\mathcal{R}, \Sigma)$, where $\mathcal{R}$ is a set of runs and Σ is a* partial *function mapping points in $\mathcal{R} \times \mathbb{N}$ to states for ψ such that the following hold for all runs $r \in \mathcal{R}$:*

1. *If $\Sigma(r, n)$ is defined, then $\Sigma(r, n')$ is defined for all $n' > n$, and $\Sigma(r, n)$, $\Sigma(r, n+1), \ldots$ is an acceptable $\rightarrow$-sequence.*
2. *For all points $(r, n) \sim_i (r', n')$, if $\Sigma(r, n)$ is defined, then $\Sigma(r', n')$ is defined and $\Sigma(r, n) \approx_i \Sigma(r', n')$.*
3. *If $\Sigma(r, n)$ and s are σ-states such that $\Sigma(r, n) \approx_i s$, then there exists a point (r', n') such that $(r, n) \sim_i (r', n')$ and $\Sigma(r', n') = s$.*
4. *If $C\varphi \in cl_0(\psi)$ and $\Sigma(r, n) \Vdash \neg C\varphi$, then there exists a point (r', n') reachable from (r, n) such that $\Sigma(r', n') \Vdash \neg\varphi$.*

An enriched$^+$ system *for ψ is a pair $(\mathcal{R}, \Sigma)$ satisfying conditions* 1, 2, *and the following modification of* 3*:*

3′. *If $\Sigma(r, n)$ is a σ-state and s is a $(\sigma \# i)$-state such that $\Sigma(r, n) \approx_i s$, then there exists a point (r', n') such that $(r, n) \sim_i (r', n')$ and $\Sigma(r', n') = s$.*

Intuitively, in an enriched (resp., enriched$^+$) system, the points where Σ is defined are the points that are "relevant" to the truth of certain formulas at certain points.

Given an enriched (resp., enriched$^+$) system $(\mathcal{R}, \Sigma)$, we obtain an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ by defining the valuation π on basic propositions p by $\pi(r, n)(p) =$ **true** just when $\Sigma(r, n)$ is defined and $\Sigma(r, n) \Vdash p$.[5] The following theorem gives a sufficient condition for a formula in the basic closure to hold at a point in this standard system. If σ is the index $i_1 \ldots i_k$, let $K_\sigma\varphi$ be an abbreviation for $K_{i_1} \ldots K_{i_k}\varphi$. (If $\sigma = \epsilon$, then we take $K_\sigma\varphi$ to be φ.)

THEOREM 4.10.

(a) *If $(\mathcal{R}, \Sigma)$ is an enriched system for ψ, $\mathcal{I}$ is the associated interpreted system, φ is in the basic closure $cl_0(\psi)$, and $\Sigma(r, n)$ is defined, then $(\mathcal{I}, r, n) \vDash \varphi$ iff $\Sigma(r, n) \Vdash \varphi$.*

(b) *If $(\mathcal{R}, \Sigma)$ is an enriched$^+$ system for $\psi \in KL_m$, $\mathcal{I}$ is the associated standard system, φ is in the basic closure $cl_0(\psi)$, $\Sigma(r, n)$ is a σ-state, and $ad(K_\sigma\varphi) \leq d$, then $(\mathcal{I}, r, n) \vDash \varphi$ iff $\Sigma(r, n) \Vdash \varphi$.*

Proof. We first prove part (a). We proceed by induction on the complexity of φ. If φ is a propositional constant, then the result is immediate from the definition of $\mathcal{I}$. The cases where φ is of the form $\neg\varphi_1$ or $\varphi_1 \wedge \varphi_2$ are similarly trivial. This leaves five cases.

Case 1. Suppose that φ is of the form $\bigcirc\varphi_1$. Then $(\mathcal{I}, r, n) \vDash \varphi$ iff $(\mathcal{I}, r, n+1) \vDash \varphi_1$. Note that $\Sigma(r, n+1)$ must be defined by condition 1 of Definition 4.9. Since φ_1 is a subformula of φ it is in $cl_0(\psi)$, so it follows by the induction hypothesis that $(\mathcal{I}, r, n+1) \vDash \varphi_1$ holds precisely when $\Sigma(r, n+1) \Vdash \varphi_1$. By condition 1, $\Sigma(r, n) \rightarrow \Sigma(r, n+1)$, so we obtain from Lemma 4.6(a) that $\Sigma(r, n+1) \Vdash \varphi_1$ iff $\Sigma(r, n) \Vdash \bigcirc\varphi_1$. Putting the pieces together, we get $(\mathcal{I}, r, n) \vDash \varphi$ iff $\Sigma(r, n) \Vdash \varphi$.

Case 2. Suppose that φ is of the form $\varphi_1 \, U \, \varphi_2$. Then the subformulas φ_1 and φ_2 are also in $cl_0(\psi)$. Note also that by condition 1 of Definition 4.9, $\Sigma(r, n')$ is defined for all $n' \geq n$, and $\Sigma(r, n), \Sigma(r, n+1), \ldots$ is an acceptable $\rightarrow$-sequence. Thus, if $\Sigma(r, n) \Vdash \varphi_1 U \varphi_2$, then, by the definition of acceptability, there exists some $n' \geq n$ such

[5] This definition makes p false at points where Σ is undefined. We could just as well have made p true at such points, without changing our results.

that $\Sigma(r,n') \Vdash \varphi_2$ and $\Sigma(r,k) \Vdash \varphi_1$ for $n \le k < n'$. By the induction hypothesis, this implies that $(\mathcal{I},r,n') \models \varphi_2$ and $(\mathcal{I},r,k) \models \varphi_1$ for $n \le k < n'$. In other words, we have $(\mathcal{I},r,n) \models \varphi_1 \, U \, \varphi_2$. Conversely, if $(\mathcal{I},r,n) \models \varphi_1 \, U \, \varphi_2$, then by the induction hypothesis and the semantics of U we have that there exists some $n' \ge n$ such that $\Sigma(r,n') \Vdash \varphi_2$ and $\Sigma(r,k) \Vdash \varphi_1$ for $n \le k < n'$. Since $\Sigma(r,n) \to \Sigma(r,n+1) \to \cdots \to \Sigma(r,n')$, it follows using Lemma 4.6(c) that $\Sigma(r,n) \Vdash \varphi_1 \, U \, \varphi_2$.

Case 3. Suppose that φ is of the form $K_i\varphi_1$. We first show that $\Sigma(r,n) \Vdash K_i\varphi_1$ implies $(\mathcal{I},r,n) \models K_i\varphi_1$. Assume $\Sigma(r,n) \Vdash K_i\varphi_1$ and suppose that $(r,n) \sim_i (r',n')$. Then by condition 2 of Definition 4.9, we have that $\Sigma(r',n')$ is defined and $\Sigma(r,n) \approx_i \Sigma(r',n')$. Since $K_i\varphi \in cl_0(\psi)$ we obtain $\Sigma(r,n') \Vdash K_i\varphi_1$. By K3 this implies $\Sigma(r,n') \Vdash \varphi_1$. Since $\varphi \in cl_0(\psi)$, by the induction hypothesis, we obtain that $(\mathcal{I},r',n') \models \varphi_1$. This shows that $(\mathcal{I},r',n') \models \varphi_1$ for all points $(r',n') \sim_i (r,n)$. That is, we have $(\mathcal{I},r,n) \models K_i\varphi_1$.

For the converse, suppose that $\Sigma(r,n) \Vdash \neg K_i\varphi_1$ and that $\Sigma(r,n)$ is a σ-state. By Lemma 4.6(b), there exists a σ-state t such that $\Sigma(r,n) \approx_i t$ and $t \Vdash \neg\varphi_1$. By condition 3 of Definition 4.9, there exists a point (r',n') such that $(r,n) \sim_i (r',n')$ and $\Sigma(r',n') = t$. Using the induction hypothesis we obtain that $(\mathcal{I},r',n') \models \neg\varphi_1$. It follows that $(\mathcal{I},r,n) \models \neg K_i\varphi_1$.

Case 4. If φ is of the form $E\varphi_1$, the result follows easily from the induction hypothesis, using axiom C1.

Case 5. Suppose that φ is of the form $C\varphi_1$. By condition 2 of Definition 4.9 we have that $\Sigma(r',n')$ is defined for all (r',n') reachable from (r,n). An easy induction on the length of the path from (r,n) to (r',n'), using the fact that $K_iC\varphi_1$ is in the basic closure and axioms C1, C2, and K3, can be used to show that $\Sigma(r',n') \Vdash C\varphi_1$ for each point (r',n') reachable from (r',n). Using C1, C2, and K3, it is easy to see that $\Sigma(r',n') \Vdash \varphi_1$. By the induction hypothesis, this implies that $(\mathcal{I},r',n') \models \varphi_1$. Thus, $(\mathcal{I},r,n) \models C\varphi_1$.

For the converse, suppose that $\Sigma(r,n) \Vdash \neg C\varphi_1$. Then by condition 4 of Definition 4.9, we have $\Sigma(r',n') \Vdash \neg\varphi_1$ for some point (r',n') reachable from (r,n). By the induction hypothesis, we have that $(\mathcal{I},r',n') \models \neg\varphi_1$, and hence $(\mathcal{I},r,n) \models \neg C\varphi_1$.

For part (b), since $\psi \in KL_m$, we only need to check the analogues of Cases 1, 2, and 3 above. The proofs in Cases 1 and 2 are identical to those above. The proof of Case 3 is also quite similar, but we must be a little careful in applying the inductive hypothesis. So suppose that φ is of the form $K_i\varphi_1$, $\Sigma(r,n)$ is a σ-state, and $ad(K_\sigma\varphi) \le d$. The implication from left to right, showing that if $\Sigma(r,n) \Vdash K_i\varphi$, then $(\mathcal{I},r,n) \models K_i\varphi$, is identical to that above. We just need the observation that if $(r,n) \sim_i (r',n')$, then $\Sigma(r',n')$ is a τ-state, where $\tau\#i = \sigma\#i$. It follows that $ad(K_\tau\varphi) \le ad(K_{\sigma\#i}\varphi) \le ad(K_\sigma K_i\varphi) \le d$, so we can apply the inductive hypothesis to conclude that $(\mathcal{I},r',n') \models \varphi_1$. For the converse, the proof is again similar. Note that if $ad(K_\sigma K_i\varphi) \le d$, then $|\sigma\#i| \le d$, so by Lemma 4.6(b), there exists a $(\sigma\#i)$-state t such that $\Sigma(r,n) \approx_i t$ and $t \Vdash \neg\varphi_1$. By condition 3′ of Definition 4.9, there exists a point (r',n') such that $(r,n) \sim_i (r',n')$ and $\Sigma(r',n') = t$. Since $ad(K_{\sigma\#i}\varphi) = ad(K_\sigma K_i\varphi) \le d$, using the induction hypothesis we obtain that $(\mathcal{I},r',n') \models \neg\varphi_1$. It follows that $(\mathcal{I},r,n) \models \neg K_i\varphi_1$. □

COROLLARY 4.11. *If $(\mathcal{R},\Sigma)$ is an enriched (resp., enriched$^+$) system for ψ, $\mathcal{I}$ is the associated interpreted system, and (r,n) is a point of $\mathcal{I}$ such that $\Sigma(r,n)$ is an ϵ-state and $\Sigma(r,n) \Vdash \psi$, then $(\mathcal{I},r,n) \models \psi$.*

We apply this corollary in all our completeness proofs, constructing an appropriate enriched or enriched$^+$ system in all cases.

5. Proofs of soundness and completeness. We are now in a position to prove the completeness results claimed in section 3. Sections 5.1–5.3 will deal with the cases involving only perfect recall, synchrony, and unique initial states. The cases involving no learning are a little more complex and are dealt with in sections 5.4–5.7.

5.1. Dealing with $\mathcal{C}_m$, $\mathcal{C}_m^{sync}$, $\mathcal{C}_m^{uis}$, and $\mathcal{C}_m^{sync,uis}$ (Theorems 3.1 and 3.2). The fact that $\mathrm{S5C}_m^U$ is sound for $\mathcal{C}_m$, the class of all systems, is straightforward and left to the reader (see also [1]). To prove completeness of $\mathrm{S5}_m^U$ for the language KL_m and of $\mathrm{S5C}_m^U$ for the language CKL_m with respect to $\mathcal{C}_m$, $\mathcal{C}_m^{sync}$, $\mathcal{C}_m^{uis}$, and $\mathcal{C}_m^{sync,uis}$, we construct an enriched system and use Corollary 4.11. The proof proceeds in the same way whether or not common knowledge is in the language. We assume here that the language includes common knowledge and that we are dealing with the axiom system $\mathrm{S5C}_m^U$ when constructing the states in the enriched structure. Recall that in this case we work with ϵ-states only.

The following result suffices for the generation of the acceptable sequences required for the construction of an enriched system in the cases not involving no learning; a more complex construction will be required in the presence of no learning.

LEMMA 5.1. *Every finite $\rightarrow$-sequence of states can be extended to an infinite acceptable sequence.*

Proof. First note that for every σ-state s there exists a state t with $s \rightarrow t$, for otherwise, $s \Vdash \neg\bigcirc\varphi_t$ for all σ-states t, which contradicts $\vdash \bigcirc \bigvee_{t \text{ a } \sigma\text{-state}} \varphi_t$. (Note that $\vdash \bigcirc \bigvee_{t \text{ a } \sigma\text{-state}} \varphi_t$ follows from Lemma 4.1(c) and RT1.) Thus every finite sequence of states can be extended to an infinite sequence, and it remains to show that the obligations arising from the until formulas can be satisfied.

Suppose that the finite $\rightarrow$-sequence is $s_0 \rightarrow \cdots \rightarrow s_n$, where $s_k = (\sigma, X_k)$ for $k = 1, \ldots, n$. Now, for any formula $\varphi_1 \, U \, \varphi_2 \in X_0$, it follows using T3 and the fact that the s_i directly decide each of the formulas φ_1, φ_2, and $\varphi_1 \, U \, \varphi_2$ that either the obligation imposed by $\varphi_1 \, U \, \varphi_2$ at s_0 is already satisfied in the sequence $(s_0, \ldots, s_n)$, or else $s_n \Vdash \varphi_1 \, U \, \varphi_2$ and $s_k \Vdash \varphi_1$ for $0 \leq k \leq n$. In the latter case, by Lemma 4.6(c), there exists a sequence $s_n \rightarrow s_{n+1} \rightarrow \cdots \rightarrow s_{n'}$ such that $s_{n'} \Vdash \varphi_2$ and $s_k \Vdash \varphi_1$ for $n \leq k < n'$. This gives a finite extension of the original sequence that satisfies the obligation imposed by $\varphi_1 \, U \, \varphi_2$ at s_0. Applying this argument to the remaining obligations at s_0, we eventually obtain a finite sequence that satisfies all the obligations at s_0. We may then move on to s_1 and apply the same procedure. It is clear that in the limit we obtain an acceptable sequence extending the original sequence. □

For each agent i, let O_i be the function that maps the state (σ, U) to the pair $(\sigma \# i, U/K_i)$. O_i is also used later in our other constructions. Given a state s, we call $O_i(s)$ agent i's *current information* at s. Let x be a new object not equal to any state. We say that a sequence $S = (x, x, \ldots, x, s_N, s_{N+1}, \ldots)$ is an *acceptable sequence from N* if it starts with N copies of x and the suffix $(s_N, s_{N+1}, \ldots)$ is an acceptable $\rightarrow$-sequence of states for ψ. Given a sequence S acceptable from N, we define a run r as follows. For each agent i, take $r_i(n) = (n, S)$ when $n < N$, and take $r_i(n) = (n, O_i(s_n))$ otherwise. For the environment component e, take $r_e(n) = S_n$ (so that $r_e(n) = x$ if $n < N$ and $r_e(n) = s_n$ for $n \geq N$).

Let $\mathcal{R}^{sync}$ be the set of all runs so obtained, and define the partial function Σ on points in $\mathcal{R}^{sync} \times \mathbb{N}$ so that $\Sigma(r, n) = s_n$ when r is derived from a sequence $(x, x, \ldots, x, s_N, s_{N+1}, \ldots)$ acceptable from N and $n \geq N$, and $\Sigma(r, n)$ is undefined otherwise.

LEMMA 5.2. *The pair $(\mathcal{R}^{sync}, \Sigma)$ is an enriched system.*

Proof. It is immediate from the construction that $(\mathcal{R}^{sync}, \Sigma)$ satisfies conditions

1 and 2 of Definition 4.9. To see that it satisfies condition 3, suppose that (r, n) is a point such that $\Sigma(r, n)$ is defined and $\Sigma(r, n) \approx_i s$. By Lemma 5.1 there exists an acceptable sequence $(s_n, s_{n+1}, \ldots)$ with $s = s_n$. Let r' be the run obtained from the sequence $(x, \ldots, x, s_n, s_{n+1}, \ldots)$. Then it is immediate that $(r, n) \sim_i (r', n')$ and $\Sigma(r', n') = s$. Finally, to see that it satisfies condition 4, suppose that $C\varphi \in cl_0(\psi)$ and $\Sigma(r, n) \Vdash \neg C\varphi$. By Lemma 4.8, there is a state t reachable from $\Sigma(r, n)$ through the relations $\approx_i$ such that $t \Vdash \neg\varphi$. An easy inductive argument on the length of the path from $\Sigma(r, n)$ to t, using condition 3, shows that there is a point (r', n') reachable from (r, n) through the relations $\sim_i$ such that $\Sigma(r', n') = t$. Thus, the enriched system satisfies condition 4. □

Clearly the system $\mathcal{R}^{sync}$ is synchronous, so the interpreted system $\mathcal{I}$ derived from $(\mathcal{R}^{sync}, \Sigma)$ is also synchronous. Let s be an ϵ-state such that $s \Vdash \psi$. Such a state must exist because ψ was assumed consistent. By Lemma 5.1 there exists an acceptable sequence $(s_0, s_1, \ldots)$ with $s = s_0$. Let r be the corresponding run in $\mathcal{R}^{sync}$. Corollary 4.11 implies that $(\mathcal{I}, r, 0) \models \psi$. This establishes the completeness of the axiomatization $\mathrm{S5C}_m^U$ for the language CKL_m (resp., of $\mathrm{S5}_m^U$ for the language KL_m) with respect to the classes of systems $\mathcal{C}_m$ and $\mathcal{C}_m^{sync}$. To establish completeness of these axiomatizations for the corresponding languages with respect to the classes of systems $\mathcal{C}_m^{uis}$ and $\mathcal{C}_m^{sync,uis}$, we make use of the following result, which shows that sound and complete axiomatizations for the class of systems satisfying some subset of the properties of perfect recall and synchrony are also sound and complete axiomatizations for the class of systems with the same subset of these properties, but with unique initial states in addition. This completes the proofs of Theorems 3.1 and 3.2.

LEMMA 5.3. *Suppose x is a subset of $\{pr, sync\}$. If $\varphi \in CKL_m$ is satisfiable with respect to $\mathcal{C}_m^x$, then it is also satisfiable with respect to $\mathcal{C}_m^{x,uis}$.*

Proof. Suppose $\mathcal{I} = (\mathcal{R}, \pi) \in \mathcal{C}_m^x$. We define a system $\mathcal{I}'$ by adding a new initial state to each run in $\mathcal{R}$. Formally, we define the system $\mathcal{I}' = (\mathcal{R}', \pi')$ as follows. Let l be some local state that does not occur in $\mathcal{I}$ and let s_e be any state of the environment. For each run $r \in \mathcal{R}$, let r^+ be the run such that $r^+(0) = (s_e, l, \ldots, l)$ and $r^+(n+1) = r(n)$. Let $\mathcal{R}' = \{r^+ : r \in \mathcal{R}\}$. The valuation π' is given by $\pi'(r, 0)(p) = \mathbf{false}$ and $\pi'(r, n+1)(p) = \pi(r, n)(p)$ for $n \geq 0$ and propositions p. It is clear that $\mathcal{I}'$ is a system with unique initial states. Moreover, if $\mathcal{I}$ is synchronous, then so is $\mathcal{I}'$, and if $\mathcal{I}$ is a system with perfect recall, then so is $\mathcal{I}'$. A straightforward induction on the construction of the formula $\varphi \in CKL_m$ now shows that, for all points (r, n) in $\mathcal{I}$, we have $(\mathcal{I}, r, n) \models \varphi$ iff $(\mathcal{I}', r^+, n+1) \models \varphi$. □

5.2. Dealing with $\mathcal{C}_m^{pr}$ and $\mathcal{C}_m^{pr,uis}$ (Theorem 3.5). We want to show that $\mathrm{S5}_m^U + \mathrm{KT3}$ is sound and complete with respect to $\mathcal{C}_m^{pr}$. We first consider soundness. As we observed above, all axioms and rules of inference other than KT3 are known to be sound in *all* systems, so their soundness in systems $\mathcal{C}_m^{pr}$ is immediate. The next result establishes soundness of KT3.

LEMMA 5.4. *All instances of* KT3 *are valid in $\mathcal{C}_m^{pr}$.*

Proof. To show that KT3 is sound, we assume that $(\mathcal{I}, r, n) \models K_i\varphi_1 \wedge \bigcirc(K_i\varphi_2 \wedge \neg K_i\varphi_3)$. We show that $(\mathcal{I}, r, n) \models L_i((K_i\varphi_1)\, U\, [(K_i\varphi_2)\, U\, \neg\varphi_3])$. Now it follows from the assumption that $(\mathcal{I}, r, n+1) \models \neg K_i\varphi_3$, so there exists a point (r', n') such that $(r, n+1) \sim_i (r', n')$ and $(\mathcal{I}, r', n') \models \neg\varphi_3$. Since $\mathcal{I} \in \mathcal{C}_m^{pr}$, by Lemma 2.2(d), either (i) $(r, n) \sim_i (r', n')$ or (ii) there exists a number $l < n'$ such that $(r, n) \sim_i (r', l)$ and $(r, n+1) \sim_i (r', k)$ for all k with $l < k \leq n'$. We claim that in either case $(\mathcal{I}, r, n) \models L_i((K_i\varphi_1)\, U\, [(K_i\varphi_2)\, U\, \neg\varphi_3])$. In case (i), since $(\mathcal{I}, r', n') \models \neg\varphi_3$, we have $(\mathcal{I}, r', n') \models (K_i\varphi_1)\, U\, [(K_i\varphi_2)\, U\, \neg\varphi_3]$. The desired conclusion is then immediate from

the fact that $(r,n) \sim_i (r',n')$. In case (ii), since $(\mathcal{I},r,n) \models K_i\varphi_1$, and $(r,n) \sim_i (r',l)$, we have that $(\mathcal{I},r',l) \models K_i\varphi_1$. Similarly, because $(\mathcal{I},r,n+1) \models K_i\varphi_2$, we obtain that $(\mathcal{I},r',k) \models K_i\varphi_2$ for all k with $l < k \leq n'$. Together with $(\mathcal{I},r',n') \models \neg\varphi_3$, this implies that $(\mathcal{I},r',l) \models (K_i\varphi_1)\, U\, [(K_i\varphi_2)\, U\, \neg\varphi_3]$. Again, since $(r,n) \sim_i (r',l)$, we obtain that $(\mathcal{I},r,n) \models L_i((K_i\varphi_1)\, U\, [(K_i\varphi_2)\, U\, \neg\varphi_3])$. □

We now establish a lemma characterizing the interaction of knowledge and time in the premodel. This result will enable us to satisfy the perfect-recall requirement in using the premodel to construct an interpreted system. It is convenient to introduce the notation $[s]_i$, where s is a state, for the set of $(\sigma\#i)$-states t such that $s \approx_i t$. The reader is encouraged to compare the following result with Lemma 2.2(c).

LEMMA 5.5. *Suppose that the axiomatization includes* KT3. *Then for all σ-states s,t and for all $(\sigma\#i)$-states t', if $s \to t$ and $t \approx_i t'$, then either* (a) $s \approx_i t'$ *or* (b) *there exists a $(\sigma\#i)$-state s' such that $s \approx_i s'$ and there exists a sequence of $(\sigma\#i)$-states $u_0 \to u_1 \to \cdots \to u_n = t'$, where $n \geq 0$, such that $s' \to u_0$ and $u_l \approx_i u_{l+1}$ for all $l = 0,\ldots,n-1$.*

Proof. We derive a contradiction from the assumption that $s \to t$ and $t \approx_i t'$, but $s \not\approx_i t'$ and for all $(\sigma\#i)$-states s' such that $s \approx_i s'$ and all sequences of $(\sigma\#i)$-states $u_0 \to u_1 \to \cdots \to u_n$ such that $s' \to u_0$ and $u_i \approx_i u_{i+1}$ for $i = 0,\ldots,n-1$, we have $u_n \neq t'$. Let T be the smallest set of $(\sigma\#i)$-states such that

1. if $v \in [s]_i$, $v \to v'$, and $v' \in [t]_i$, then $v' \in T$; and
2. if $v \in T$, $v \to v'$, and $v' \in [t]_i$, then $v' \in T$.

Because $s \not\approx_i t'$, it follows from the fact that $\approx_i$ is an equivalence relation that the intersection $[s]_i \cap [t]_i$ is empty. Additionally, t' is not in T, for otherwise we could find a sequence of the sort presumed not to exist. Thus, for all $v \in T$, we have $\vdash \varphi_v \Rightarrow \neg\varphi_{t'}$. This implies that $\vdash \varphi_T \Rightarrow \neg\varphi_{t'}$. Let T' be the set of $(\sigma\#i)$-states v' such that $v \to v'$ for some $v \in T$. We want to show that

$$v' \Vdash \varphi_T \vee (\neg K_i\Phi^+_{t,i} \wedge \neg\varphi_{t'}) \tag{5.1}$$

for all $v' \in T'$. If $v' \in T$, then clearly we have $v' \Vdash \varphi_T$, so (5.1) holds. If $v' \notin T$, then the second condition in the definition of T implies that v' is not in $[t]_i$. It follows using Lemma 4.3(d) that $v' \Vdash \neg K_i\Phi^+_{t,i}$. Further, $t \not\approx_i v'$ implies that $v' \neq t'$, so $v' \Vdash \neg\varphi_{t'}$. Thus, again we have (5.1). Since (5.1) holds for all $v' \in T$, it follows that $\vdash \varphi_{T'} \Rightarrow (\varphi_T \vee (\neg K_i\Phi^+_{t,i} \wedge \neg\varphi_{t'}))$. Now by Lemma 4.4, we have $\vdash \varphi_T \Rightarrow \bigcirc\varphi_{T'}$, so using T1 and RT1 we obtain that $\vdash \varphi_T \Rightarrow \bigcirc(\varphi_T \vee (\neg K_i\Phi^+_{t,i} \wedge \neg\varphi_{t'}))$. Combining this with $\vdash \varphi_T \Rightarrow \neg\varphi_{t'}$ and using Lemma 4.5, we get that $\vdash \varphi_T \Rightarrow \neg(K_i\Phi^+_{t,i}\, U\, \varphi_{t'})$. In particular, we obtain $v \Vdash \neg(K_i\Phi^+_{t,i}\, U\, \varphi_{t'})$ for all states v in T.

We now repeat this argument to obtain a similar conclusion for the elements of $[s]_i$. Since t' is not in $[s]_i$ we have that $v \in [s]_i$ implies $v \Vdash \neg\varphi_{t'}$. Further, since $[s]_i \cap [t]_i$ is empty we also have by Lemma 4.3(d) that $v \in [s]_i$ implies $v \Vdash \neg K_i\Phi^+_{t,i}$. Using T3 this yields that $\vdash \Phi^+_{s,i} \Rightarrow \neg(K_i\Phi^+_{t,i}\, U\, \varphi_{t'})$.

Let P be the set of $(\sigma\#i)$-states v' such that $v \to v'$ for some $v \in [s]_i$. Let $v' \in P$. We want to show that

$$v' \Vdash \Phi^+_{s,i} \vee (\neg K_i\Phi^+_{s,i} \wedge \neg(K_i\Phi^+_{t,i}\, U\, \varphi_{t'})). \tag{5.2}$$

If $v' \in [s]_i$, then clearly $v' \Vdash \Phi^+_{s,i}$, so (5.2) holds. If $v' \notin [s]_i$, then, by Lemma 4.3(d), we have that $v' \Vdash \neg K_i\Phi^+_{s,i}$. We now consider two subcases: (a) $v' \in T$ and (b) $v' \notin T$. If $v' \in T$, then, as we showed earlier, we have $v' \Vdash \neg(K_i\Phi^+_{t,i} U \varphi_{t'})$. If $v' \notin T$, then by the definition of T it follows that $t \not\approx_i v'$. By Lemma 4.3(d), this implies that $v' \Vdash \neg K_i\Phi^+_{t,i}$.

Further, since $t \approx_i t'$, we also obtain that $v' \neq t'$, so $v' \Vdash \neg\varphi_{t'}$. Using T3, this yields $v' \Vdash \neg(K_i\Phi^+_{t,i}\, U\, \varphi_{t'})$, and again we have (5.2). Using Lemma 4.4, we obtain that $\vdash \Phi^+_{s,i} \Rightarrow \bigcirc[\Phi^+_{s,i} \vee (\neg K_i\Phi^+_{s,i} \wedge \neg(K_i\Phi^+_{t,i}\, U\, \varphi_{t'}))]$. Applying Lemma 4.5 to this and the result of the preceding paragraph establishes that $\vdash \Phi^+_{s,i} \Rightarrow \neg(K_i\Phi^+_{s,i} U\, (K_i\Phi^+_{t,i} U\, \varphi_{t'}))$.

It follows using Lemma 4.3(b), R2, and K2 that $s \Vdash K_i\neg(K_i\Phi^+_{s,i} U\, (K_i\Phi^+_{t,i} U\, \varphi_{t'}))$. By KT3, we obtain $s \Vdash \neg(K_i\Phi^+_{s,i} \wedge \bigcirc(K_i\Phi^+_{t,i} \wedge L_i\varphi_{t'}))$. Since, by Lemma 4.3(b), $s \Vdash K_i\Phi^+_{s,i}$, we obtain using T2 that $s \Vdash \bigcirc\neg(K_i\Phi^+_{t,i} \wedge L_i\varphi_{t'})$. Because $s \to t$, we have that $\varphi_s \wedge \bigcirc\varphi_t$ is consistent, so it follows that $\varphi_t \wedge \neg(K_i\Phi^+_{t,i} \wedge L_i\varphi_{t'})$ is consistent. However, by Lemma 4.3, $t \Vdash K_i\Phi^+_{t,i} \wedge L_i\varphi_{t'}$, so this is a contradiction. □

We are now ready to define, for each consistent ψ, an enriched$^+$ system for ψ that establishes completeness of $\mathrm{S5}^U_m + \mathrm{KT3}$ with respect to $\mathcal{C}^{pr}_m$. The runs of this system are those derived from the acceptable sequences $(s_0, s_1, \ldots)$ (of states for ψ) by putting $r_e(n) = s_n$ and $r_i(n) = O_i(s_0)\# \cdots \# O_i(s_n)$, for each agent i and $n \geq 0$. Thus, $r_i(n)$ is the sequence of current information that agent i has had up to time n. Let $\mathcal{R}^{pr}$ be the set of runs defined in this way. The function Σ is given by $\Sigma(r, n) = s_n$ for each $n \geq 0$.

LEMMA 5.6. *Suppose that the axiomatization includes* KT3. *Then* $(\mathcal{R}^{pr}, \Sigma)$ *is an enriched*$^+$ *system.*

Proof. It is clear that $(\mathcal{R}^{pr}, \Sigma)$ satisfies conditions 1 and 2 of Definition 4.9. It remains to show that condition 3′ holds. So suppose that $\Sigma(r, n)$ is a σ-state and $\Sigma(r, n) \approx_i s$ for some $(\sigma\#i)$-state s. We must find a point (r', n') such that $\Sigma(r', n') = s$.

The proof proceeds by induction on n. The result for $n = 0$ is immediate, since we can take r' to be an acceptable sequence starting from s (such a sequence exists by Lemma 5.1), so $\Sigma(r', 0) = s$ and clearly $(r, 0) \sim_i (r', 0)$.

Now suppose that $n > 0$ and the result holds for $n - 1$. Because $\Sigma(r, n-1) \to \Sigma(r, n)$ and $\Sigma(r, n) \approx_i s$, it follows by Lemma 5.5 that either (a) $\Sigma(r, n-1) \approx_i s$ or (b) there exists a $(\sigma\#i)$-state s' such that $\Sigma(r, n-1) \approx_i s'$ and there exists a sequence of $(\sigma\#i)$-states $u_0 \to u_1 \to \cdots \to u_k$ such that $s' \to u_0$, $u_l \approx_i u_{l+1}$ for $l = 0, \ldots, k-1$, and $u_k = s$. By the induction hypothesis, there exists for every $(\sigma\#i)$-state t with $\Sigma(r, n-1) \approx_i t$ a point (r', n') such that $(r, n-1) \sim_i (r', n')$ and $\Sigma(r', n') = t$. In case (a), we take $t = s$, and we then have that $\Sigma(r, n-1) \approx_i \Sigma(r, n)$ and $\Sigma(r', n') = s$. It follows that $(r, n) \sim_i (r', n-1)$, and by the transitivity of $\sim_i$, we also have $(r, n) \sim_i (r', n')$. Hence we are done. In case (b), we take $t = s'$. Suppose that r' is derived from the sequence $(v_0, v_1, \ldots)$. Let r'' be any run derived from an acceptable sequence with initial segment $(v_0, \ldots, v_{n'}, u_0, \ldots, u_k)$. Again, such a run exists by Lemma 5.1. By construction, $\Sigma(r'', n' + k + 1) = u_k = s$. Moreover, since $r''_i(n') = r'_i(n') = r_i(n-1)$ and $O_i(u_l) = O_i(s)$ for all $l = 0, \ldots, k$, we have $r''_i(n' + k + 1) = r''_i(n')\#O_i(u_0)\# \cdots \#O_i(u_k) = r_i(n-1)\#O_i(s) = r_i(n)$, and hence $(r, n) \sim_i (r'', n' + k + 1)$. □

Now take any ϵ-state s such that $s \Vdash \psi$, and let r be a run derived from an acceptable sequence starting with s. By construction, the system $\mathcal{I}$ obtained from the enriched$^+$ system is in $\mathcal{C}^{pr}_m$, and by Corollary 4.11, we have $(\mathcal{I}, r, 0) \vDash \psi$. Thus, ψ is satisfiable in $\mathcal{C}^{pr}_m$. By Lemma 5.3, ψ is also satisfiable in systems in $\mathcal{C}^{pr,uis}_m$. Since this argument applies to an arbitrary formula ψ consistent with respect to $\mathrm{S5}^U_m + \mathrm{KT3}$, this completes the proof of Theorem 3.5.

5.3. Dealing with $\mathcal{C}^{pr,sync}_m$ and $\mathcal{C}^{pr,sync,uis}_m$ (Theorem 3.6). We now show that $\mathrm{S5}^U_m + \mathrm{KT2}$ is sound and complete with respect to KL_m for the classes of systems

$\mathcal{C}_m^{pr,sync}$ and $\mathcal{C}_m^{pr,sync,uis}$. For soundness, the following result suffices.

LEMMA 5.7. *All instances of* KT2 *are valid in* $\mathcal{C}_m^{pr,sync}$.

Proof. Let $\mathcal{I}$ be a system in $\mathcal{C}_m^{pr,sync}$ and let r be a run of $\mathcal{I}$. Suppose that $(\mathcal{I}, r, n) \vDash K_i \bigcirc \varphi$. If $(r, n+1) \sim_i (r', n')$, then by synchrony we must have $n' = n+1$. Thus, by perfect recall and synchrony, we have $(r, n) \sim_i (r', n'-1)$. It follows that $(\mathcal{I}, r', n'-1) \vDash \bigcirc\varphi$, which implies that $(\mathcal{I}, r', n') \vDash \varphi$. This shows that $(\mathcal{I}, r', n') \vDash \varphi$ for all $(r', n') \sim_i (r, n+1)$. Thus, we have $(\mathcal{I}, r, n+1) \vDash K_i\varphi$, and hence $(\mathcal{I}, r, n) \vDash \bigcirc K_i\varphi$. □

Before constructing an enriched$^+$ system for the completeness proof, we first note a property of the premodel, analogous to that given in Lemma 5.5.

LEMMA 5.8. *Suppose that the axiomatization includes* KT2. *Then for all* σ*-states* s, t *with* $s \to t$, *we have that for all* $(\sigma\#i)$*-states* t' *with* $t \approx_i t'$ *there exists a* $(\sigma\#i)$*-state* s' *such that* $s \approx_i s'$ *and* $s' \to t'$.

Proof. By way of contradiction, suppose that s, t are σ-states with $s \to t$, that t' is a $(\sigma\#i)$-state such that $t \approx_i t'$, but that for all $(\sigma\#i)$-states s' such that $s \approx_i s'$, we have that $s' \Vdash \neg\bigcirc\varphi_{t'}$. By T2, we have that $s' \Vdash \bigcirc\neg\varphi_{t'}$ for all $(\sigma\#i)$-states s' such that $s \approx_i s'$. By Lemma 4.3(b), it follows that $s \Vdash K_i\bigcirc\neg\varphi_{t'}$. By KT2, we have that $s \Vdash \bigcirc K_i\neg\varphi_{t'}$. Since $s \to t$, it follows that $\varphi_t \wedge K_i\neg\varphi_{t'}$ is consistent. However, since $t \approx_i t'$, by Lemma 4.3(c), we have $t \Vdash L_i\varphi_{t'}$. This is a contradiction. □

To construct the enriched$^+$ system, we now take $\mathcal{R}^{pr,sync}$ to be the set of runs r derived from acceptable sequences $(s_0, s_1, \ldots)$ of states for the formula ψ by putting $r_e(n) = s_n$ and $r_i(n) = O_i(s_0)\ldots O_i(s_n)$, for each agent i and $n \geq 0$. The notation $O_i(s_0)\ldots O_i(s_n)$ is meant to denote the sequence formed by concatenating $O_i(s_0), O_i(s_1), \ldots, O_i(s_n)$. Thus, the length of the sequence is $n+1$, which enforces synchrony. Again, the function Σ is given by $\Sigma(r, n) = s_n$ for each $n \geq 0$.

LEMMA 5.9. *Suppose that the axiomatization includes* KT2. *Then* $(\mathcal{R}^{pr,sync}, \Sigma)$ *is an enriched*$^+$ *system.*

Proof. Conditions 1 and 2 of the definition of an enriched$^+$ system are immediate. To show that condition 3′ holds, suppose that $\Sigma(r, n)$ is a σ-state and that t is a $(\sigma\#i)$-state such that $\Sigma(r, n) \approx_i t$. Suppose that r is derived from the acceptable sequence $(s_0, s_1, \ldots)$, so $\Sigma(r, n) = s_n$. It follows from Lemma 5.8 that there exists a $\to$-sequence $t_0 \to \cdots \to t_n$ such that $t_n = t$ and $s_j \approx_i t_j$ for $j = 1, \ldots, n$. By Lemma 5.1, this sequence may be extended to an infinite acceptable sequence. Taking r' to be the run derived from this sequence, we see that $(r, n) \sim_i (r', n)$ and $\Sigma(r', n) = t$. □

Take $\mathcal{I}^{pr,sync}$ to be the system obtained from $(\mathcal{R}^{pr,sync}, \Sigma)$. By construction, this system is in $\mathcal{C}^{pr,sync}$. Now take any ϵ-state s such that $s \Vdash \psi$, and let r be a run derived from an acceptable sequence starting with s. By construction, the system $\mathcal{I}$ obtained from the enriched$^+$ system is in $\mathcal{C}_m^{pr,sync}$, and by Corollary 4.11, we have $(\mathcal{I}, r, 0) \vDash \psi$. Thus, ψ is satisfiable in $\mathcal{C}_m^{pr,sync}$. By Lemma 5.3, ψ is also satisfiable in systems in $\mathcal{C}_m^{pr,sync,uis}$. Since this argument applies to any formula ψ consistent with respect to $\mathrm{S5}_m^U + \mathrm{KT2}$, this completes the proof of Theorem 3.6.

5.4. Dealing with $\mathcal{C}_m^{nl}$ (Theorem 3.7). We want to show that $\mathrm{S5}_m^U + \mathrm{KT4}$ is sound and complete for KL_m with respect to $\mathcal{C}_m^{nl}$. For soundness, it suffices to show that KT4 is valid in $\mathcal{C}_m^{nl}$. This is straightforward.

LEMMA 5.10. *All instances of* KT4 *are valid in* $\mathcal{C}_m^{nl}$.

Proof. Suppose that $\mathcal{I} \in \mathcal{C}_m^{nl}$ and $(\mathcal{I}, r, n) \vDash K_i\varphi\, U\, K_i\psi$. We want to show that $(\mathcal{I}, r, n) \vDash K_i(K_i\varphi\, U\, K_i\psi)$. Thus, if $(r', n') \sim_i (r, n)$, we must show that $(\mathcal{I}, r', n') \vDash K_i\varphi\, U\, K_i\psi$. Since $(\mathcal{I}, r, n) \vDash K_i\varphi\, U\, K_i\psi$, there exists $l \geq n$ such that $(\mathcal{I}, r, l) \vDash K_i\psi$ and $(\mathcal{I}, r, k) \vDash K_i\varphi \wedge \neg K_i\psi$ for all k with $n \leq k < l$. Note that this means that if

$n \leq k < l$, then $r_i(k) \neq r_i(l)$. Since $\mathcal{I} \in \mathcal{C}_m^{nl}$ and $(r,n) \sim_i (r',n')$, there must be some $l' \geq n'$ such that $((r,n),\ldots,(r,l))$ is $\sim_i$-concordant with $((r,n'),\ldots,(r,l'))$. Thus, there exists some h, a partition $S_1,\ldots,S_h$ of the sequence $((r,n),\ldots,(r,l))$, and a partition $T_1,\ldots,T_h$ of the sequence $((r',n'),\ldots,(r,l'))$ such that for all $j = 1,\ldots,h$, we have $(r,k) \sim_i (r',k')$ for all points $(r,k) \in S_j$ and $(r',k') \in T_j$. It easily follows that $(\mathcal{I},r',n') \models K_i\varphi\, U\, K_i\psi$, as desired. □

For completeness, we define an appropriate enriched$^+$ system. As we shall see, the demands of no learning make this a little more subtle than in the case of no forgetting.

For the remainder of this section, consistency and provability are with respect to a logic that includes $\mathrm{S5}_m^U + \mathrm{KT4}$. Fix a consistent formula ψ such that $ad(\psi) = d$.

Our first step is to prove an analogue of Lemma 5.5.

LEMMA 5.11. *Suppose that the axiomatization includes* KT4. *If s is a σ-state, t is a $(\sigma\#i)$-state, $s \approx_i t$, and $s \to s'$, then there exists a sequence $t_0,\ldots,t_k$ such that* (a) $t = t_0$, (b) $t_j \approx_i s$ *for* $j < k$, (c) $t_j \to t_{j+1}$ *for* $j < k$, *and* (d) $s' \approx_i t_k$.

Proof. If $s' \approx_i t$, then we can take the sequence to consist only of t, and we are done. Otherwise, since $\varphi_s \wedge \bigcirc\varphi_{s'}$ is consistent, it follows from Lemma 4.3(b) that $\varphi_s \wedge K_i\Phi_{s,i}^+\, U\, K_i\Phi_{s',i}^+$ is consistent. Moreover, by Lemma 4.3(c), we have that $\varphi_s \Vdash L_i\varphi_t$. Thus, $\varphi_s \wedge L_i\varphi_t \wedge K_i\Phi_{s,i}^+\, U\, K_i\Phi_{s',i}^+$ is consistent. Using KT4, it follows that $\varphi_t \wedge K_i\Phi_{s,i}^+ U\, K_i\Phi_{s',i}^+$ is consistent. The result now follows from Lemma 4.7. □

Unfortunately, Lemma 5.11 does not suffice to construct an enriched$^+$ system. Roughly speaking, the problem is the following. In the case of perfect recall, we used Lemma 5.5 to show that, given a $\to$-sequence $S = (s_0,\ldots,s_n)$ of σ-states and a $(\sigma\#i)$-state t such that $s_n \approx_i t$, we can construct a $\to$-sequence T of $(\sigma\#i)$-states ending with t such that S is $\approx_i$-concordant with T. There is no problem then extending T to an acceptable sequence. Moreover, we can extend S and T independently to acceptable sequences; all that matters is that the finite prefixes of these sequences—namely, S and T—are $\approx_i$-concordant. With no learning, on the other hand, it is the infinite suffixes that must be $\approx_i$-concordant. Given a $\to$-sequence $S = (s_0,\ldots)$ of σ-states and a $(\sigma\#i)$-state t such that $s_0 \approx_i t$, using Lemma 5.11, we can find a $\to$-sequence T starting with t that is $\approx_i$-concordant with S. This suggests that it is possible to find the appropriate sequences for the construction of runs satisfying the no learning condition. Unfortunately, it does not follow from the acceptability of S that T is also acceptable. This makes it necessary to work with a smaller set of sequences than the set of all acceptable sequences, and to build up the sequences S and T simultaneously. To ensure that the appropriate obligations are satisfied at all points in the set of runs constructed, we need to work not just with single states, but with trees of states.

A *k-tree for ψ* (with $k \leq d$) is a set S of states with index σ such that $|\sigma| \leq k$, containing a unique ϵ state such that if $s \in S$ is a σ-state, then

- if t is a $(\sigma\#i)$-state such that $s \approx_i t$ and $|\sigma\#i| \leq k$, then $t \in S$;
- if $\sigma = \tau\#i$, then there is a τ-state t in S such that $s \approx_i t$.

We extend the $\to$ relation to k-trees as follows. If S_1 and S_2 are k-trees for ψ, then $S_1 \to_f S_2$ if f is a function associating with each σ-state $s \in S_1$ a finite sequence of σ-states in $S_1 \cup S_2$ such that

- if $f(s) = (s_0,\ldots,s_k)$, then
  - $s = s_0$,
  - $s_0 \to \cdots \to s_k$,
  - $s_0,\ldots,s_{k-1} \in S_1$ and $s_k \in S_2$;
- if $s \approx_i s'$, then $f(s)$ and $f(s')$ are $\approx_i$-concordant;

- for at least one $s \in S_1$, the sequence $f(s)$ has length at least 2.

Given two sequences of σ-states $\alpha = (s_0, \ldots, s_k)$ and $\beta = (t_0, \ldots)$, where α is finite, the *fusion* of α and β, denoted $\alpha \cdot \beta$, is defined only if $s_k = t_0$; in this case, it is the sequence $(s_0, s_{k-1}, t_0, \ldots)$. Given an infinite sequence $\mathcal{S} = S_0 \rightarrow_{f_0} S_1 \rightarrow_{f_1} S_2 \rightarrow_{f_2} \cdots$ of k-trees, we say a sequence α of σ-states is *compatible with* $\mathcal{S}$ if there exists some h, and σ-states $s_h, s_{h+1}, \ldots$ with $s_j \in S_j$ for $j \geq h$, such that $\alpha = f_h(s_h) \cdot f_{h+1}(s_{h+1}) \cdot \ldots$. (Implicit in this notation is the assumption that this fusion product is defined, so that the last state in $f_j(s_j)$ is the same as the first state in $f_{j+1}(s_{j+1})$ for $j \geq h$.) $\mathcal{S}$ is *acceptable* if every $\rightarrow$-sequence compatible with $\mathcal{S}$ is infinite and acceptable. Our goal is to construct an acceptable sequence of d-trees; we then use this to define the enriched$^+$ system.

Note that by Lemma 4.3, the formula φ_s essentially describes the subtree below s of any k-tree containing s. Given a k-tree S and a σ-state s in S, we inductively define a formula $tree_{S,s}$ that describes all of S from the point of view of s. If s is an ϵ-state, then $tree_{S,s} = \varphi_s$. Otherwise, if s is a $(\tau \# i)$-state, where $\tau \neq \tau \# i$, then

$$tree_{S,s} = \varphi_s \wedge \bigwedge_{\{\tau\text{-states } t\colon\ s \approx_i t\}} L_i tree_{S,t}.$$

If S and T are k-trees, $s \in S$, and $t \in T$, then we write $(S, s) \rightarrow^+ (T, t)$ if there exists a sequence of k-trees $S_0, \ldots, S_l$ and functions $f_0, \ldots, f_{l-1}$ such that $S_0 \rightarrow_{f_0} \cdots \rightarrow_{f_{l-1}} S_l$, $S_0 = S$, $S_l = T$, $f_j(s) = (s)$ for $j \leq l-2$, and $f_{l-1}(s) = (s, t)$.

LEMMA 5.12. *Suppose that the axiomatization includes* KT4*, S is a k-tree, and s is a σ-state in S, where $|\sigma| = k$.*

(a) *If t is a σ-state and $tree_{S,s} \wedge \bigcirc(\varphi_t \wedge \xi)$ is consistent, then there exists a k-tree T such that $t \in T$, $(S, s) \rightarrow^+ (T, t)$, and $tree_{T,t} \wedge \xi$ is consistent.*

(b) *$tree_{S,s} \Rightarrow \bigcirc \bigvee_{\{(T,t)\colon\ (S,s) \rightarrow^+ (T,t)\}} tree_{T,t}$ is provable.*

(c) *If $tree_{S,s} \wedge \varphi\ U\ \varphi'$ is consistent, then for some $l \geq 0$ there is a sequence $S_0, \ldots, S_l$ of k-trees and states $s_0, \ldots, s_l$ such that* (i) *$s_j \in S_j$,* (ii) *$(S, s) = (S_0, s_0)$,* (iii) *$(S_j, s_j) \rightarrow^+ (S_{j+1}, s_{j+1})$ for $j = 0, \ldots, l-1$,* (iv) *$tree_{S_j,s_j} \wedge \varphi$ is consistent for $j = 0, \ldots, l-1$, and* (v) *$tree_{S_l,s_l} \wedge \varphi'$ is consistent.*

Proof. We proceed by induction on k. The case that $k = 0$ is immediate using standard arguments, since then $tree_{S,s}$ is just φ_s.

So suppose $k > 0$ and $\sigma = \tau \# i$, with $\sigma \neq \tau$. We first prove part (a) in the case that ξ is of the form $K_i \xi'$, then part (b), the general case of part (a), and part (c). First consider (a) in the case that ξ is of the form $K_i \xi'$. Note that $tree_{S,s} \wedge \bigcirc(\varphi_t \wedge K_i \xi')$ implies $tree_{S,s} \wedge K_i \Phi_{s,i}\ U\ K_i(\xi' \wedge \Phi_{t,i})$. From the definition of k-tree, it follows that there is a τ-state s' in S such that $s \approx_i s'$. Let S' be the $(k-1)$-tree consisting of all σ'-states in S with $|\sigma'| \leq k-1$. From KT4, it follows that

$$tree_{S',s'} \wedge K_i \Phi_{s,i}\ U\ K_i(\xi' \wedge \Phi_{t,i})$$

is consistent. Applying part (c) of the inductive hypothesis, we get a sequence $S_0, \ldots, S_l$ of $(k-1)$-trees and states $s_0, \ldots, s_l$ such that (i) $s_j \in S_j$, (ii) $(S', s') = (S_0, s_0)$, (iii) $(S_j, s_j) \rightarrow^+ (S_{j+1}, s_{j+1})$ for $j = 0, \ldots, l-1$, (iv) $tree_{S_j,s_j} \wedge K_i \Phi_{s,i}$ is consistent for $j = 0, \ldots, l-1$, and (v) $tree_{S_l,s_l} \wedge K_i(\xi' \wedge \Phi_{t,i})$ is consistent. It follows by definition that there is a sequence $T_0, \ldots, T_m$ of $(k-1)$-trees and functions $f_0, \ldots, f_{m-1}$ such that $T_0 \rightarrow_{f_0} \rightarrow \cdots \rightarrow_{f_{m-1}} T_m$, $T_0 = S_0$, and $T_m = S_l$. Moreover, there are elements $t_0, \ldots, t_m$ such that $t_0 = s'$, $t_m = s_l$, if $j < m$, then $t_j = s_{j'}$ for some $j' \leq j$, and if $t_j = t_{j+1}$, then $f_j(t_j) = (t_j)$, while if $t_j \neq t_{j+1}$, then $f_j(t_j) = (t_j, t_{j+1})$ for $j = 0, \ldots, m-1$.

Let T'_j be the unique k-tree extending T_j for $j = 0, \ldots, m$. Since $\varphi_{t_j} \wedge K_i \Phi_{s,i}$ is consistent for $j < m$, we have that $t_j \approx_i s$, and so $s \in T'_j$ for $j < m$. Similarly, we have that $t \in T'_m$. We now show how to construct f'_j for $j < m$. For each state $u' \in T'_j - T_j$, there must exist a state $u \in T_j$ and an agent j' such that $u \approx_{j'} u'$. (There may be more than one such state u, of course. In this case, in the construction below, we can pick u arbitrarily.) It easily follows from Lemma 5.11 that there exists a sequence $\alpha_{u'}$ starting with u' that is $\approx_{j'}$-concordant with $f_j(u)$. Moreover, we can take $\alpha_{t_j} = (s)$ for $j < m-1$, and take $\alpha_{t_{m-1}} = (s,t)$. We define f'_j so that it agrees with f_j on T_j, and for each $u' \in T'_j - T_j$, we have $f'_j(u') = \alpha_{u'}$.

Notice that $T'_0 = S$. If $m > 0$, it follows immediately from the definition that $(S,s) \rightarrow^+ (T_m, t)$ and that $tree_{T_m,t} \wedge K_i \xi'$ is consistent. If $m = 0$, it is easy to check that we must have $t \in S$, for we have $s' \approx_i t$. Since we also have $s' \approx_i s$, it follows that $s \approx_i t$. Define f so that $f(u) = (u)$ for $u \neq s$ and $f(s) = (s,t)$. Then $(S,s) \rightarrow_f (S,t)$. Since $s \rightarrow t$, we have $(S,s) \rightarrow^+ (S,t)$. This completes the proof of part (a).

To prove part (b), suppose not. Then $tree_{S,s} \wedge \bigcirc \bigwedge_{\{(T,t):\ (S,s) \rightarrow^+ (T,t)\}} \neg tree_{T,t}$ is consistent. Straightforward temporal reasoning shows that there must be some u such that

$$tree_{S,s} \wedge \bigcirc \left(\varphi_u \wedge \bigwedge_{\{(T,t):\ (S,s) \rightarrow^+ (T,t)\}} \neg tree_{T,t} \right) \tag{5.3}$$

is consistent. Now $\neg tree_{T,t}$ is equivalent to $\neg \varphi_t \vee \bigvee_{\{\tau\text{-states } t':\ t' \approx_i t\}} K_i \neg tree_{T,t'}$. Thus, it follows that the consistency of (5.3) implies that for each tree T such that $(S,s) \rightarrow^+ (T,u)$, there is a τ-state $t_T \approx_i u$ such that

$$tree_{S,s} \wedge \bigcirc \left(\varphi_u \wedge K_i \left(\bigwedge_{\{T:\ (S,s) \rightarrow^+ (T,u)\}} \neg tree_{T,t_T} \right) \right) \tag{5.4}$$

is consistent. By part (a), there exists a k-tree T' and $t' \in T'$ such that $(S,s) \rightarrow^+ (T',t')$ and $tree_{T',t'} \wedge \varphi_u \wedge K_i(\bigwedge_{\{T:\ (S,s) \rightarrow^+ (T,u)\}} \neg tree_{T,t_T})$ is consistent. But this means that $t' = u$. Thus, we have a contradiction, since $tree_{T',u} \wedge K_i \neg tree_{T',t_{T'}}$ is inconsistent.

The general case of part (a) follows easily from part (b). Part (c) also follows from part (b), using arguments much like those of Lemma 4.7; we omit details here. □

LEMMA 5.13. *Suppose that the axiomatization includes* KT4 *and ψ is consistent. Then there is an acceptable sequence of d-trees such that ψ is true at the root of the first tree.*

Proof. The key part of the proof is to show that given a finite sequence $S_0 \rightarrow_{f_0} \cdots \rightarrow_{f_{l-1}} S_l$ of d-trees and a σ-state s in S_l such that $s \Vdash \bigcirc \varphi$ (resp., $s \Vdash \varphi_1 \, U \, \varphi_2$), we can extend the sequence of trees in such a way as to satisfy this obligation. This follows easily from Lemmas 5.11 and 5.12. In more detail, suppose $s \Vdash \varphi_1 U \varphi_2$. Let S' consist of all τ-states in S_l, with $|\tau| \leq k = |\sigma|$. By Lemma 5.12, we can find a sequence of k-trees starting with S' that satisfies this obligation. Using Lemma 5.11, we can extend this to a sequence of d-trees starting with S_l that satisfies the obligation. The argument in the case that $s_k \Vdash \bigcirc \varphi$ is similar. We can then take care of the obligations one by one, and construct an acceptable sequence, in the obvious way.

Since ψ is consistent, there must be some tree S with root s_0 such that $s_0 \Vdash \psi$. We just extend S as above to complete the proof. □

Once we have an acceptable sequence $\mathcal{S}$ of d-trees as in Lemma 5.13, we can easily construct the enriched$^+$ system much as we did in the case of perfect recall.

Given a $\rightarrow$-sequence $s_0 \rightarrow s_1 \rightarrow \cdots$ the *nl-run* r derived from it is defined so that $r_e(n) = s_n$ and $r_i(n) = O_i(s_n)\#O_i(s_{n+1})\#\cdots$ for each agent i and $n \geq 0$. Thus, while for perfect recall we take $r_i(n)$ to consist of the agent's current information up to time n, for no learning we take $r_i(n)$ to consist of the current information from time n on. The construction stresses the duality between perfect recall and no learning. Let $\mathcal{R}^{nl}$ consist of all the *nl*-runs derived from $\rightarrow$-sequences that are compatible with $\mathcal{S}$. Again, the function Σ is given by $\Sigma(r, n) = s_n$ for each $n \geq 0$.

LEMMA 5.14. *Suppose that the axiomatization includes* KT4. *Then* $(\mathcal{R}^{nl}, \Sigma)$ *is an enriched*$^+$ *system.*

Proof. Again, conditions 1 and 2 in the definition of enriched$^+$ system follow immediately from the construction. For condition 3′, suppose that (r, n) is a point in the system, $\Sigma(r, n)$ is a σ-state, and s is a $(\sigma\#i)$-state with $s \approx_i \Sigma(r, n)$. Suppose that $\mathcal{S} = S_0 \rightarrow_{f_0} S_1 \rightarrow_{f_1} S_2 \rightarrow_{f_2} \cdots$. By definition, the *nl*-run r is derived from some sequence $(s_0, s_1, s_2, \ldots)$ of σ-states compatible with $\mathcal{S}$. Suppose that $\Sigma(r, n)$ is in the interval of this sequence from S_k. Then s must also be in S_k. Let $(t_0, t_1, \ldots)$ be the (unique) sequence compatible with $\mathcal{S}$ that starts at s in S_k. Let r' be the run derived from this sequence. Then, by definition, we have $\Sigma(r', 0) = s$ and $(r, n) \sim_i (r', 0)$. □

Take $\mathcal{I}^{nl}$ to be the system obtained from $(\mathcal{R}^{nl}, \Sigma)$. By construction, this system is in $\mathcal{C}^{nl}$. Now take any ϵ-state s such that $s \Vdash \psi$, and let r be a run derived from a sequence compatible with $\mathcal{S}$ starting with s. It follows that $(\mathcal{I}, r, 0) \vDash \psi$. Thus, ψ is satisfiable in $\mathcal{C}^{nl}_m$. This completes the proof of Theorem 3.7.

5.5. Dealing with $\mathcal{C}^{nl,pr}_m$ and $\mathcal{C}^{nl,pr,uis}_1$ (Theorem 3.8). We now want to show that $\mathrm{S5}^U_m + \mathrm{KT3} + \mathrm{KT4}$ is sound and complete for KL_m with respect to $\mathcal{C}^{nl,pr}_m$. Soundness is immediate from Lemmas 5.4 and 5.10.

For completeness, we construct an enriched$^+$ system much as in the proof of Theorem 3.7, using k-trees. By Lemma 5.13, there is an acceptable sequence $\mathcal{S}$ of d-trees such that ψ is true at the root of the first tree. Given a $\rightarrow$-sequence $(s_0, s_1, \ldots)$, the *nl–pr-run* r derived from it is defined so that $r_e(n) = s_n$ and $r_i(n) = (O_i(s_0)\#O_i(s_1)\#\cdots\#O_i(s_n), O_i(s_n)\#O_i(s_{n+1})\#\cdots)$ for each agent i and $n \geq 0$. Thus, the agents' local states enforce both perfect recall (by keeping track of all the information up to time n) and no learning (by keeping track of the current information from time n on). Let $\mathcal{R}^{nl,pr}$ consist of all *nl–pr*-runs that are derived from $\rightarrow$-sequences that have a suffix that is compatible with $\mathcal{S}$. Note that now we consider $\rightarrow$-sequences whose *suffixes* are compatible with $\mathcal{S}$. The reason we have allowed the greater generality of suffixes will become clear shortly. Since $\mathcal{S}$ is acceptable, it is easy to see that every such $\rightarrow$-sequence must be infinite and acceptable. Again, the function Σ is given by $\Sigma(r, n) = s_n$ for each $n \geq 0$.

LEMMA 5.15. *Suppose that the axiomatization includes* KT3 *and* KT4. *Then* $(\mathcal{R}^{nl,pr}, \Sigma)$ *is an enriched*$^+$ *system.*

Proof. As usual, conditions 1 and 2 in the definition of enriched$^+$ system follow immediately from the construction. For condition 3′, suppose that (r, n) is a point in the system, $\Sigma(r, n)$ is a σ-state, and s is a $(\sigma\#i)$-state. Suppose that $\mathcal{S} = S_0 \rightarrow_{f_0} S_1 \rightarrow_{f_1} S_2 \rightarrow_{f_2} \cdots$. By definition, the *nl–pr*-run r is derived from a $\rightarrow$-sequence $(s_0, s_1, \ldots)$ that has a suffix $(s_N, s_{N+1}, \ldots)$ that is compatible with $\mathcal{S}$, and $\Sigma(r, n) = s_n$. There are now two cases to consider. If $n \geq N$, then there exists some k such that s_n is in S_k. Then s must also be in S_k. Let $(u_0, u_1, \ldots)$ be the unique sequence compatible with $\mathcal{S}$ that starts with s in S_k. By Lemma 5.5, there exist a sequence $v_0 \rightarrow \cdots \rightarrow v_h$ of $\sigma\#i$-states such that $v_h = s$ and $(v_0, \ldots, v_h)$ is $\approx_i$-compatible

with $(s_0, \ldots, s_n)$. Consider the $\rightarrow$-sequence formed from the fusion of $(v_0, \ldots, v_h)$ and $(u_0, u_1, \ldots)$. By construction, the *nl–pr*-run r' derived from this $\rightarrow$-sequence is in $\mathcal{R}^{nl,pr}$, $\Sigma(r', h) = s$, and $(r, n) \sim_i (r', h)$.

Now suppose $n < N$. By Lemma 5.5, there exist $(\sigma\#i)$-states $v_0 \rightarrow \cdots \rightarrow v_h$ such that $v_h = s$ and $(v_0, \ldots, v_h)$ is $\approx_i$-concordant with $(s_0, \ldots, s_n)$. Moreover, by Lemma 5.11, this sequence can be extended to a sequence $(v_0, \ldots, v_k)$ that is $\approx_i$-concordant with $(s_0, \ldots, s_N)$. Since $s_N \in S_M$ for some M, we must have $v_k \in S_M$. Let $(u_0, u_1, \ldots)$ be the unique sequence compatible with $\mathcal{S}$ that starts at v_k in S_M. Consider the $\rightarrow$-sequence formed from the fusion of $(v_0, \ldots, v_k)$ and $(u_0, u_1, \ldots)$. By construction, the *nl–pr*-run r' derived from this $\rightarrow$-sequence is in $\mathcal{R}^{nl,pr}$, $\Sigma(r', h) = s$, and $(r, n) \sim_i (r', h)$. □

Again, we complete the proof by taking $\mathcal{I}^{nl,pr}$ to be the system obtained from $(\mathcal{R}^{nl,pr}, \Sigma)$. By construction, this system is in $\mathcal{C}^{nl,pr}$ and satisfies ψ. This shows that $\mathrm{S5}_m^U + \mathrm{KT3} + \mathrm{KT4}$ is a sound and complete axiomatization for the language KL_m with respect to $\mathcal{C}_m^{nl,pr}$.

The fact that it is also a sound and complete axiomatization for the language KL_1 with respect to $\mathcal{C}_1^{nl,pr,uis}$ follows immediately from the following lemma.

LEMMA 5.16. *The formula $\psi \in KL_1$ is satisfiable with respect to $\mathcal{C}_1^{nl,pr}$ (resp., $\mathcal{C}_1^{nl,pr,sync}$) iff it is satisfiable with respect to $\mathcal{C}_1^{nl,pr,uis}$ (resp., $\mathcal{C}_1^{nl,pr,sync,uis}$).*

Proof. Clearly if ψ is satisfiable with respect to $\mathcal{C}_1^{nl,pr,uis}$ (resp., $\mathcal{C}_1^{nl,pr,sync,uis}$) it is satisfiable with respect to $\mathcal{C}_1^{nl,pr}$ (resp., $\mathcal{C}_1^{nl,pr,sync}$). For the converse, suppose that $(\mathcal{I}, r^*, n^*) \models \psi$, where $\mathcal{I} = (\mathcal{R}, \pi) \in \mathcal{C}_1^{nl,pr}$. For each run $r \in \mathcal{R}$, define the run r^+ just as in Lemma 5.3, to be the result of adding a new initial state to r. Let $\mathcal{R}' = \{r^+ : (r, 0) \sim_1 (r^*, 0)\}$. Define π' as on $\mathcal{R}'$ as in Lemma 5.3, so that $\pi'(r^+, 0)(p) = \mathbf{false}$ for all primitive propositions p, and $\pi'(r^+, n+1) = \pi(r, n)$. Let $\mathcal{I}' = (\mathcal{R}', \pi')$. Clearly $\mathcal{I}' \in \mathcal{C}_1^{nl,pr,uis}$, and if $\mathcal{I}$ is synchronous, then so is $\mathcal{I}'$. We claim that $(\mathcal{I}', r^+, n+1) \models \varphi$ iff $(\mathcal{I}, r, n) \models \varphi$ for all formulas $\varphi \in KL_1$ and all $r^+ \in \mathcal{R}'$. We prove this by induction on the structure of φ. The only nontrivial case is if φ is of the form $K_1\varphi'$. But this case is immediate from the observation that if $r^+ \in \mathcal{R}'$, $r' \in \mathcal{R}$, and $(r, n) \sim_1 (r', n')$, then since $\mathcal{I}$ is a system of perfect recall, we must have $(r, 0) \sim_1 (r', 0)$, and hence $(r')^+ \in \mathcal{R}'$. We leave details of the proof of the claim to the reader. From the claim, it follows that ψ is satisfiable in $\mathcal{C}_1^{nl,pr,uis}$, and that if ψ is satisfiable in $\mathcal{C}_1^{nl,pr,sync}$, then it is also satisfiable in $\mathcal{C}_1^{nl,pr,sync,uis}$. □

5.6. Dealing with $\mathcal{C}_m^{nl,sync}$ (Theorem 3.9). We now want to show that $\mathrm{S5}_m^U + \mathrm{KT5}$ is sound and complete for KL_m with respect to $\mathcal{C}_m^{nl,sync}$. Soundness follows from the following lemma.

LEMMA 5.17. *All instances of* KT5 *are valid in $\mathcal{C}_m^{nl,sync}$.*

Proof. Suppose that $\mathcal{I} \in \mathcal{C}_m^{nl,sync}$ and $(\mathcal{I}, r, n) \models \bigcirc K_i\varphi$. We want to show that $(\mathcal{I}, r, n) \models K_i\bigcirc\varphi$. Thus, suppose that $(r', n') \sim_i (r, n)$. We must show that $(\mathcal{I}, r', n') \models \bigcirc\varphi$. By synchrony, we must have $n' = n$. Moreover, by no learning and synchrony, we have that $(r, n+1) \sim_i (r', n+1)$. Since $(\mathcal{I}, r, n+1) \models K_i\varphi$, it follows that $(\mathcal{I}, r', n+1) \models \varphi$, and hence that $(\mathcal{I}, r', n) \models \bigcirc\varphi$, as desired. □

For completeness, we construct an enriched$^+$ system much as in the proof of Theorem 3.7, using k-trees, with an appropriate strengthening of the $\rightarrow$ relation.

We start by proving the following analogue of Lemma 5.11.

LEMMA 5.18. *Suppose that the axiomatization includes* KT5. *If s is a σ-state, t is a $(\sigma\#i)$-state, $s \approx_i t$, and $s \rightarrow s'$, then there exists a $(\sigma\#i)$-state t' such that $t \rightarrow t'$ and $s' \approx_i t'$.*

Proof. Since $\varphi_s \wedge \bigcirc\varphi_{s'}$ is consistent, it follows from Lemma 4.3(b) that $\varphi_{s_0} \wedge \bigcirc K_i\Phi^+_{s',i}$ is consistent. Moreover, by Lemma 4.3(c), we have that $\varphi_s \Vdash L_i\varphi_t$. Thus, $\varphi_s \wedge L_i\varphi_t \wedge \bigcirc K_i\Phi^+_{s',i}$ is consistent. Using KT5, it follows that $\varphi_s \wedge L_i\varphi_t \wedge K_i\bigcirc\Phi^+_{s',i}$ is consistent. It follows that $\varphi_t \wedge \bigcirc\Phi^+_{s',i}$ is consistent. Hence, there is some $(\sigma\#i)$-state t' such that $s' \approx_i t'$ and $\varphi_t \wedge \bigcirc\varphi_{t'}$ is consistent. Thus, we have $s' \approx_i t'$ and $t \to t'$. □

If S and T are k-trees, $s \in S$, and $t \in T$, we define $(S,s) \rightarrow^{sync,+} (T,t)$ if $S \rightarrow_f T$ for some f such that $f(s) = (s,t)$ and $f(s')$ has length 2 for all $s' \in S$. We now get the following simplification of Lemma 5.12.

LEMMA 5.19. *Suppose that the axiomatization includes* KT5*, S is a k-tree, and s is a σ-state in S, where $|\sigma| = k$.*

(a) *If $tree_{S,s} \wedge \bigcirc(\varphi_t \wedge \xi)$ is consistent, then there exists a k-tree T and $t \in T$ such that $(S,s) \rightarrow^{sync,+} (T,t)$ and $tree_{T,t} \wedge \xi$ is consistent.*

(b) *$tree_{S,s} \Rightarrow \bigcirc \bigvee_{\{(T,t):\ (S,s) \rightarrow^{sync,+} (T,t)\}} tree_{T,t}$ is provable.*

(c) *If $tree_{S,s} \wedge \varphi\ U\ \varphi'$ is consistent, then there is a sequence $S_0, \ldots, S_l$ of k-trees and states $s_0, \ldots, s_l$ such that* (i) *$s_j \in S_j$,* (ii) *$(S,s) = (S_0,s_0)$,* (iii) *$(S_j,s_j) \rightarrow^{sync,+} (S_{j+1},s_{j+1})$ for $j = 0, \ldots, l-1$,* (iv) *$tree_{S_j,s_j} \wedge \varphi$ is consistent for $j = 0, \ldots, l-1$, and* (v) *$tree_{S_l,s_l} \wedge \varphi'$ is consistent.*

Proof. The proof is like that of Lemma 5.12, using Lemma 5.18 instead of Lemma 5.11. We leave details to the reader. □

We can then define a *sync*-acceptable sequence of trees by replacing $\rightarrow^+$ by $\rightarrow^{sync,+}$ in the definition of acceptable sequence of trees. Using Lemma 5.19, we can show that if the axiom system contains KT5, then we can construct an infinite *sync*-acceptable sequence $\mathcal{S} = S_0 \rightarrow^{sync,+} S_1 \rightarrow^{sync,+} S_2 \rightarrow^{sync,+} \cdots$ of d-trees. As in section 5.1, we use an object x not equal to any state. Given a $\rightarrow$-sequence $s_N \rightarrow s_{N+1} \rightarrow \cdots$ starting at $s_N \in S_N$ the *nl–sync-run* r derived from it is defined so that $r_e(n) = s_n$ for $n \geq N$, else $r_e(n) = x$, and for each agent i, if $n \geq N$, then $r_i(n) = (n, O_i(s_n)O_i(s_{n+1})\ldots)$, else $r_i(n) = (n, x^{N-n}O_i(s_n)O_i(s_{n+1})\ldots)$. Thus, the local state of the agent enforces synchrony (by encoding the time) and enforces no learning. Let $\mathcal{R}^{nl,sync}$ consist of all *nl–sync*-runs derived from $\rightarrow$-sequences compatible with $\mathcal{S}$, and define Σ by taking $\Sigma(r,n) = s_n$ for $n \geq N$ and $\Sigma(r,n)$ undefined for $n < N$.

LEMMA 5.20. *Suppose that the axiomatization includes* KT5*. Then $(\mathcal{R}^{nl,sync}, \Sigma)$ is an enriched$^+$ system.*

Proof. The proof is essentially the same as that of Lemma 5.14. We leave details to the reader. □

We complete the proof of Theorem 3.9 just as we did all the previous proofs.

5.7. Dealing with $\mathcal{C}^{nl,pr,sync}_m$ (Theorem 3.10). We now want to show that $S5^U_m$+KT2+KT5 is sound and complete for KL_m with respect to $\mathcal{C}^{nl,pr,sync}_m$. Soundness follows from Lemmas 5.7 and 5.17.

For completeness, we construct an enriched$^+$ system by combining the ideas of the proofs of Theorems 3.8 and 3.9. Using Lemma 5.19, we can show that if the axiom system contains KT5, then we can construct an infinite *sync*-acceptable sequence $\mathcal{S}$ of d-trees. Given a $\rightarrow$-sequence $s_0 \rightarrow s_1 \rightarrow \cdots$, the *nl–pr–sync-run* r derived from it is defined so that $r_e(n) = s_n$ and $r_i(n) = (O_i(s_1)\ldots O_i(s_n), O_i(s_n)O_i(s_{n+1})\ldots)$. Thus, the local state of the agent enforces both perfect recall and no learning. It also enforces synchrony, since the agent can determine n from the length of the first of the two sequences in its local state. Let $\mathcal{R}^{nl,pr,sync}$ consist of all *nl–pr–sync*-runs derived from $\rightarrow$-sequences with suffixes that are compatible with $\mathcal{S}$; again, we define Σ by

taking $\Sigma(r,n) = s_n$.

Using ideas similar to those in earlier proofs, we can now prove the following result.

LEMMA 5.21. *Suppose that the axiomatization includes* KT2 *and* KT5. *Then* $(\mathcal{R}^{nl,pr,sync}, \Sigma)$ *is an enriched*$^+$ *system.*

We complete the proof of Theorem 3.10 just as we did the earlier proofs.

5.8. Dealing with $\mathcal{C}_m^{nl,sync,uis}$ and $\mathcal{C}_m^{nl,pr,sync,uis}$ (Theorem 3.11). Finally, we want to show that $\mathrm{S5}_m^U + \mathrm{KT2} + \mathrm{KT5} + \{K_i\varphi \Leftrightarrow K_1\varphi\}$ is sound and complete for KL_m with respect to $\mathcal{C}_m^{nl,sync,uis}$ and $\mathcal{C}_m^{nl,pr,sync,uis}$. Soundness follows easily using the following result, which is Proposition 3.9 in [8] (restated using our notation).

PROPOSITION 5.22.

(a) $\mathcal{C}_m^{nl,sync,uis} = \mathcal{C}_m^{nl,pr,sync,uis}$.

(b) *Any formula* φ *in* KL_m *is equivalent in* $\mathcal{C}_m^{nl,sync,uis}$ *to the formula* φ' *that results by replacing all occurrences of* K_i, $i \geq 2$, *by* K_1.

It follows from part (a) of Proposition 5.22 that the same axioms characterize $\mathcal{C}_m^{nl,sync,uis}$ and $\mathcal{C}_m^{nl,pr,sync,uis}$. Now using Lemmas 5.7 and 5.17, the soundness of KT2 and KT5 follows. The soundness of $K_i\varphi \equiv K_1\varphi$ follows from part (b).

For completeness, using the axiom $K_i\varphi \equiv K_1\varphi$, it suffices to show the completeness of $\mathrm{S5}_1^U + \mathrm{KT2} + \mathrm{KT5}$ with respect to $\mathcal{C}_1^{nl,pr,sync,uis}$. By Theorem 3.10, this axiomatization is complete with respect to $\mathcal{C}_1^{nl,pr,sync}$. The result now follows using Lemma 5.16.

6. Remarks on no learning. We noted in section 2 that the definition of no learning adopted in this paper differs from that used in [5, 8]. We now comment on the reason for this change and the relationship between these alternative definitions of no learning.

First, recall from part (d) of Lemma 2.2 that agent i has perfect recall in system $\mathcal{R}$ iff

($*$) for all points $(r,n) \sim_i (r',n')$ in $\mathcal{R}$, if $k \leq n$, then there exists $k' \leq n'$ such that $(r,k) \sim_i (r',k')$.

Intuitively, no learning is the dual of perfect recall, so it seems reasonable to define no learning by replacing references to the past in a definition of perfect recall by references to the future. This was done in [5, 8], where the definition given for no learning was the following future time variant of condition ($*$), which we call no learning$'$, to distinguish it from our current definition: Agent i does not learn$'$ in system $\mathcal{R}$ iff

($**$) for all points $(r,n) \sim_i (r',n')$ in $\mathcal{R}$, if $k \geq n$, then there exists $k' \geq n'$ such that $(r,k) \sim_i (r',k')$.

The following lemma states a number of relations holding between condition ($**$) and the other properties we have considered in this paper.

LEMMA 6.1.

(a) *If agent* i *does not learn in system* $\mathcal{R}$, *then agent* i *does not learn*$'$ *in system* $\mathcal{R}$.

(b) *If system* $\mathcal{R}$ *is synchronous or if agent* i *has perfect recall in* $\mathcal{R}$, *then agent* i *does not learn in* $\mathcal{R}$ *iff agent* i *does not learn*$'$ *in* $\mathcal{R}$.

Proof. We first prove part (a). Suppose that agent i does not learn in $\mathcal{R}$. Assume that $(r,n) \sim_i (r',n')$ and let $k \geq n$. Since i does not learn, the future local state sequences at (r,n) and (r',n') are equal. It follows that there exists $k' \geq n'$ such that $(r,k) \sim_i (r',k')$. Thus, agent i does not learn$'$.

For part (b), it follows from part (a) that it suffices to show the implication from no learning′ to no learning. We consider the cases of synchrony and perfect recall independently. In each case, we show that if $(r,n) \sim_i (r',n')$, then there exists $k \geq n'$ such that the sequences $((r,n),(r,n+1))$ and $((r',n'),\ldots,(r',k))$ are $\sim_i$-concordant. It then follows by Lemma 2.3 that agent i does not learn.

Assume first that $\mathcal{R}$ is a synchronous system and that $(r,n) \sim_i (r',n')$. By synchrony, we must have $n = n'$. By no learning′, there exists $k \geq n$ such that $(r,n+1) \sim_i (r',k)$. By synchrony, k must equal $n+1$. It is immediate that $((r,n),(r,n+1))$ and $((r',n'),(r',n'+1))$ are $\sim_i$-concordant.

Next, assume that agent i has perfect recall in $\mathcal{R}$ and that $(r,n) \sim_i (r',n')$. By no learning′, there exists $k \geq n'$ such that $(r,n+1) \sim_i (r',k)$. By perfect recall, agent i's local state sequences $(r,n+1)$ and (r',k) are identical, as are the local state sequences at (r,n) and (r',n'). It follows that the sequences $((r,n),(r,n+1))$ and $((r',n'),\ldots,(r',k))$ are $\sim_i$-concordant. □

Thus, in the context of either synchrony or perfect recall, no learning and no learning′ are equivalent. However, in systems without synchrony or perfect recall, no learning′ is strictly weaker than no learning, as the following example shows. Consider the system $\mathcal{R} = \{r^1, r^2\}$ for a single agent, where the runs are defined by

$$r^1(n) = \begin{cases} (s_e, a) & \text{if } n = 0, \\ (s_e, b) & \text{if } n > 0 \text{ is odd}, \\ (s_e, c) & \text{if } n > 0 \text{ is even}, \end{cases}$$

where s_e is some state of the environment, and a, b, c are local states of agent 1, and similarly

$$r^2(n) = \begin{cases} (s_e, a) & \text{if } n = 0, \\ (s_e, c) & \text{if } n > 0 \text{ is odd}, \\ (s_e, b) & \text{if } n > 0 \text{ is even}. \end{cases}$$

This system clearly satisfies *uis* and condition $(**)$, so we have no learning′ (for both agents). However, agent 1's future local-state sequences from the points $(r^1, 0) \sim_1 (r^2, 0)$ are not $\sim_1$-concordant, so we do not have no learning. Thus, no learning and no learning′ are distinct in general.

This raises the question of which variant to take as the definition of no learning for the cases $\mathcal{C}^{nl}$ and $\mathcal{C}^{uis,nl}$. The origin of this notion in the literature lies in Ladner and Reif's paper [11], where it is motivated as arising in the context of blindfold games. Their logic LLP assumes perfect recall, so is not decisive on the distinction. However, it seems that the behavior in the above example is somewhat unnatural for this application, and the definition we have adopted in this paper better fits the intuition of a player in a blindfold game following a fixed linear strategy, but with some uncertainty about timing. It is such examples that in fact led us to use the current definition of no learning.

It is worth noting that the example above also shows that the axiom KT4 is not sound with respect to the class of systems satisfying $(**)$. Define the interpretation π of the propositions p and q on runs $r \in \mathcal{R}$ by $\pi(r,n)(p) = \mathbf{true}$ iff $r_1(n) = a$ and $\pi(r,n)(q) = \mathbf{true}$ iff $r_1(n) = b$. Let $\mathcal{I} = (\mathcal{R}, \pi)$. It is then readily seen that $(\mathcal{I}, r^1, 0) \models K_1 p U K_1 q$ but not $(\mathcal{I}, r^1, 0) \models K_1(K_1 p U K_1 q)$. Hence KT4 fails in this system. (This example is a future time version of an example used in [14] to show that the axiom KT1 is incomplete for systems with perfect recall.) We have not investigated the issue of axiomatization using no learning′ rather than no learning in the two cases where there

is a difference—$\mathcal{C}_m^{nl}$ and $\mathcal{C}_1^{nl,uis}$. We conjecture that, while there will be a relatively clean complete axiomatization in these cases, it will not be as elegant as the one proposed here. That is, the axiom that captures no learning will be somewhat more complicated than KT4. This conjecture is in line with our feeling that no learning is the "right" definition, not no learning$'$.

We remark that the complexity results of [5, 8, 7] are proved in the context of no learning$'$, but it is relatively straightforward to show that the same results hold if we use the definition of no learning instead.

7. Discussion. While we have looked in this paper at the effect on axiomatization of some combinations of classes of systems and language (48 in all!), there are certainly other cases of interest. One issue we have already mentioned is that of branching time versus linear time. Basing the temporal fragment of the language on branching time yields another 48 logics, whose complexity is studied in [5]. We would conjecture that the obvious translations of the axioms we have presented here deal with branching time, with similar proofs of completeness, but this remains to be verified.

It is worth remarking that our results are very sensitive to the language studied. As we have seen, the language considered in this paper is too coarse to reflect some properties of systems. In the absence of the other properties, synchrony and unique initial states do not require additional axioms. This may no longer be true for richer languages. For example, if we allow past-time operators [13], we need not only the additional axioms capturing the properties of these, but also new axioms describing the interaction of knowledge and time. Suppose that we add an operator $\ominus$ such that $(\mathcal{I}, r, n) \models \ominus\varphi$ if $n \geq 1$ and $(\mathcal{I}, r, n-1) \models \varphi$. Notice that $\neg\ominus \mathit{true}$ expresses the property "the time is 0" and $\ominus\neg\ominus \mathit{true}$ expresses the property "the time is 1." Similarly, we can inductively define formulas that express the property "the time is m" for each $m \geq 0$. If $\mathit{time} = m$ is an abbreviation for this formula, then $\mathit{time} = m \Rightarrow K_i(\mathit{time} = m)$ is valid in $\mathcal{C}^{sync}$ for each time m.

On the other hand, by adding past-time operators we can simplify the axiom for perfect recall. Introducing the operator S for "since," we may show that the formula

$$(K_i\varphi)S(K_i\psi) \Rightarrow K_i((K_i\varphi)S(K_i\psi))$$

is valid in $\mathcal{C}^{pr}$. This axiom very neatly expresses the meaning of perfect recall, and a comparison with KT4 shows clearly the sense in which perfect recall is a dual of no learning. Techniques similar to those developed in this paper may be used to prove that this axiom, together with the usual axioms for past time [13] and for knowledge, yields a complete axiomatization for $\mathcal{C}^{pr}$.

Besides changes to the language, there are also additional properties of systems worth considering. One case of interest is the class of *asynchronous message passing systems* of [1]. That extra axioms are required in such systems is known (see [1, Exercise 8.8]), but the question of complete axiomatization is still open.

REFERENCES

[1] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about Knowledge*, MIT Press, Cambridge, MA, 1995.

[2] R. Fagin, J. Y. Halpern, and M. Y. Vardi, *A model-theoretic analysis of knowledge*, J. ACM, 91 (1991), pp. 382–428.

[3] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi, *On the temporal analysis of fairness*, in Proceedings of the 7th ACM Symposium on Principles of Programming Languages, ACM Press, New York, 1980, pp. 163–173.

[4] J. Y. Halpern and Y. Moses, *A guide to completeness and complexity for modal logics of knowledge and belief*, Artificial Intelligence, 54 (1992), pp. 319–379.

[5] J. Y. Halpern and M. Y. Vardi, *The complexity of reasoning about knowledge and time*, in Proceedings of the 18th ACM Symposium on Theory of Computing, ACM Press, New York, 1986, pp. 304–315.

[6] J. Y. Halpern and M. Y. Vardi, *The complexity of reasoning about knowledge and time in asynchronous systems*, in Proceedings of the 20th ACM Symposium on Theory of Computing, ACM Press, New York, 1988, pp. 53–65.

[7] J. Y. Halpern and M. Y. Vardi, *The Complexity of Reasoning about Knowledge and Time: Synchronous Systems*, Research Report RJ 6097, IBM, 1988.

[8] J. Y. Halpern and M. Y. Vardi, *The complexity of reasoning about knowledge and time,* I: *Lower bounds*, J. Comput. System Sci., 38 (1989), pp. 195–237.

[9] J. Hintikka, *Knowledge and Belief*, Cornell University Press, Ithaca, NY, 1962.

[10] D. Kozen and R. Parikh, *An elementary proof of the completeness of PDL*, Theoret. Comput. Sci., 14 (1981), pp. 113–118.

[11] R. E. Ladner and J. H. Reif, *The logic of distributed protocols (preliminary report)*, in Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference, Morgan–Kaufmann, San Francisco, 1986, pp. 207–222.

[12] D. Lehmann, *Knowledge, common knowledge, and related puzzles*, in Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing, ACM, New York, 1984, pp. 62–67.

[13] O. Lichtenstein, A. Pnueli, and L. Zuck, *The glory of the past*, in Proceedings of the Workshop on Logics of Programs, R. Parikh, ed., Lecture Notes in Comput. Sci. 193, Springer-Verlag, Berlin, New York, 1985, pp. 196–218.

[14] R. von der Meyden, *Axioms for knowledge and time in distributed systems with perfect recall*, in Proceedings of the 9th IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 448–457.

[15] R. Parikh and R. Ramanujam, *Distributed processing and the logic of knowledge*, in Proceedings of the Workshop on Logics of Programs, R. Parikh, ed., Springer-Verlag, Berlin, 1985, pp. 256–268.

[16] A. Pnueli, *Linear and branching structures in the semantics and logics of reactive systems*, in Proceedings of the 12th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 194, Springer-Verlag, Berlin, New York, 1985, pp. 15–32.

[17] M. Sato, *A study of Kripke-style methods for some modal logics by Gentzen's sequential method*, Publ. Res. Inst. Math. Sci. (Kyoto University), 13 (1977), pp. 381–468.

[18] E. Spaan, *Nexttime is not necessary*, in Theoretical Aspects of Reasoning about Knowledge: Proceedings of the Third Conference, Morgan–Kaufmann, San Francisco, 1990, pp. 241–256.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 704–720

HARDNESS OF APPROXIMATION FOR VERTEX-CONNECTIVITY NETWORK DESIGN PROBLEMS*

GUY KORTSARZ†, ROBERT KRAUTHGAMER‡, AND JAMES R. LEE§

Abstract. In the survivable network design problem (SNDP), the goal is to find a minimum-cost spanning subgraph satisfying certain connectivity requirements. We study the vertex-connectivity variant of SNDP in which the input specifies, for each pair of vertices, a required number of vertex-disjoint paths connecting them.

We give the first strong lower bound on the approximability of SNDP, showing that the problem admits no efficient $2^{\log^{1-\epsilon} n}$ ratio approximation for any fixed $\epsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\mathrm{polylog}(n)})$. We show hardness of approximation results for some important special cases of SNDP, and we exhibit the first lower bound on the approximability of the related classical NP-hard problem of augmenting the connectivity of a graph using edges from a given set.

Key words. approximation algorithms, hardness of approximation, vertex connectivity, survivable network design, connectivity augmentation

AMS subject classifications. 68W25, 05C85, 05C40

DOI. 10.1137/S0097539702416736

1. Introduction.

A basic problem in network design is to find, in an input graph $G = (V, E)$ with nonnegative edge costs, a spanning subgraph of minimum cost that satisfies certain connectivity requirements; see, for example, the surveys [15, 24]. A fundamental problem in this area is the vertex-connectivity variant of the *survivable network design problem* (SNDP). Here, the input also specifies a *connectivity requirement* $k_{u,v}$ for every pair of vertices $\{u, v\}$, and the goal is to find a minimum-cost spanning subgraph with the property that, between every pair of vertices $\{u, v\}$, there are at least $k_{u,v}$ *vertex-disjoint* paths.

Many network design problems (including SNDP) are NP-hard, and a significant amount of research is concerned with *approximation algorithms* for these problems, i.e., polynomial-time algorithms that find a solution whose value is guaranteed to be within some factor (called the *approximation ratio*) of the optimum. A notable success is the 2-approximation of Jain [20] for the edge-connectivity version of SNDP, in which the paths are required only to be *edge-disjoint*. (See also [21, 14] for an extension to a more general version of SNDP.) However, for the vertex-connectivity variant of SNDP, no algorithm that achieves a sublinear (in $|V|$) approximation ratio has been found, despite a considerable amount of study.

This disparity between the known approximations for different variants of SNDP might suggest a lack in our understanding of vertex-connectivity network design or, perhaps, that vertex-connectivity problems are inherently more difficult to approxi-

*Received by the editors October 24, 2002; accepted for publication (in revised form) November 4, 2003; published electronically April 14, 2004.

http://www.siam.org/journals/sicomp/33-3/41673.html

†Department of Computer Sciences, Rutgers University, Camden, NJ 08102 (guyk@crab.rutgers.edu).

‡International Computer Science Institute (ICSI), Berkeley, CA 94704 and Computer Science Division, University of California, Berkeley, CA 94720 (robi@cs.berkeley.edu). Current address: IBM Almaden Research Center, San Jose, CA 95120 (robi@almaden.ibm.com). The research of this author was supported in part by NSF grants CCR-9820951 and CCR-0121555 and DARPA cooperative agreement F30602-00-2-0601.

§Computer Science Division, University of California, Berkeley, CA 94720 (jrl@cs.berkeley.edu). The research of this author was supported by NSF grant CCR-0121555.

mate. Resolving this question (see, e.g., [34, section 30.2]) is one of the important open problems in the field of approximation algorithms. We provide an answer by showing that there is a striking difference between the approximability of the edge- and vertex-connectivity variants of SNDP. Specifically, we show that it is hard to approximate the vertex-connectivity variant of SNDP within a factor of $2^{\log^{1-\epsilon}|V|}$ for any fixed $\epsilon > 0$.

In general, we address the *hardness of approximation* of vertex-connectivity problems by presenting relatively simple variants of SNDP that are nevertheless hard to approximate. Therefore, unless stated otherwise, *connectivity* means vertex-connectivity, *disjoint paths* means vertex-disjoint paths, and all graphs are assumed to be undirected. (For a more in-depth account of approximation algorithms for edge-connectivity problems, see [15, 24].)

Two special cases of SNDP for which we show hardness of approximation are the *subset connectivity problem* and the *outconnectivity to a subset problem* (OSP). In the first problem, the input contains a subset S of the vertices and a number k, and the goal is to find a minimum-cost subgraph that contains at least k vertex-disjoint paths between every pair of vertices in S. This is SNDP with $k_{u,v} = k$ for all $u, v \in S$ and $k_{u,v} = 0$ otherwise. In the second problem, the input contains a special vertex r (called the *root*), a subset S of the vertices, and a number k, and the goal is to find a minimum-cost subgraph that contains at least k vertex-disjoint paths between r and any vertex in S. In other words, this is SNDP with $k_{r,v} = k$ for all $v \in S$ and $k_{u,v} = 0$ otherwise.

A related problem is the *vertex-connectivity augmentation problem* ($\mathrm{VCAP}_{\ell,k}$), where the goal is to find a minimum-cost set of edges that augments an ℓ-connected graph into a k-connected graph. We exhibit the first hardness of approximation results for this problem.

1.1. Previous work. Throughout, let $n = |V|$ denote the number of vertices in the input graph G.

A classical and well-studied special case of SNDP is the problem of finding a minimum-cost k-vertex connected spanning subgraph, i.e., the special case where $k_{u,v} = k$ for all vertex pairs $\{u, v\}$. This is called the *k-vertex connected spanning subgraph problem* (k-VCSS). k-VCSS is NP-hard even for $k = 2$ and *uniform costs* (i.e., when all edges have the same cost), as this problem already generalizes the Hamiltonian cycle problem (note that a 2-connected subgraph of G has n edges if and only if it is a Hamiltonian cycle). By a similar argument, the outconnectivity to a subset problem is also NP-hard, even for $k = 2$ and $S = V \setminus \{r\}$ [3]. It immediately follows that SNDP (which is a more general problem) is also NP-hard. $\mathrm{VCAP}_{0,2}$ is NP-hard by a similar argument [11], and $\mathrm{VCAP}_{1,2}$ is proved to be NP-hard in [17].

Most previous work on approximating vertex-connectivity problems concentrated on upper bounds, i.e., on designing approximation algorithms. An approximation ratio of $2k$ for k-VCSS was obtained in [3] by a straightforward application of [16], and the approximation ratio was later improved to k in [28]. Recently, Cheriyan, Vempala, and Vetta [5] devised improved approximation algorithms for the problem. For the case where $k \leq \sqrt{n/6}$, they achieve approximation ratio $6H(k) = O(\log k)$, where $H(k)$ is the kth harmonic number. For the case where $k \leq (1-\epsilon)n$, they achieve approximation ratio $\sqrt{n/\epsilon}$, which was very recently improved to $O(\frac{1}{\epsilon}\log^2 k)$ by Kortsarz and Nutov [29]. (An approximation ratio of $O(\log k)$ claimed in [31] was found to be erroneous; see [32].)

Better approximation ratios are known for several special cases of k-VCSS. For

$k \leq 7$ an approximation ratio of $\lceil (k+1)/2 \rceil$ is known (see [25] for $k = 2$, [2] for $k = 2, 3$, [9] for $k = 4, 5$, and [28] for $k = 6, 7$). For *metric costs* (i.e., when the costs satisfy the triangle inequality) an approximation ratio $2 + \frac{(k-1)}{n}$ is given in [28] (building on a ratio $2 + \frac{2(k-1)}{n}$ previously shown in [25]). For uniform costs, an approximation ratio of $1 + 1/k$ is obtained in [4]. For k-VCSS in a complete Euclidean graph in $\mathbb{R}^{O(1)}$, a polynomial time approximation scheme (i.e., factor $1+\epsilon$ for any fixed $\epsilon > 0$) is devised in [7]. For 2-VCSS in dense graphs and graphs with maximum degree 3, improved approximations are given in [6].

The connectivity augmentation problem also has attracted a lot of attention. A 2-approximation for $\mathrm{VCAP}_{1,2}$ is shown in [17, 26]. In the case where every pair of vertices in the graph forms an augmenting edge of unit cost, $\mathrm{VCAP}_{k,k+1}$ is not known to be in P nor to be NP-hard. For the latter problem, a $k-2$ additive approximation is presented in [22], and optimal algorithms for small values of k are shown in [11, 35, 19, 18].

The special case of OSP with $S = V \setminus \{r\}$ (called the k-outconnectivity problem), can be approximated within ratio 2; see, for example, [25]. Approximation algorithms for related problems are given in [3].

In contrast, there are few lower bounds for approximating vertex-connectivity problems. It is shown in [7] that 2-VCSS is APX-hard (i.e., there exists some fixed $\epsilon > 0$ such that approximation within ratio $1+\epsilon$ is NP-hard) even for bounded-degree graphs with uniform costs and for complete Euclidean graphs in $\mathbb{R}^{\log n}$. In [6], APX-hardness is shown for instances of 2-VCSS on dense graphs and graphs of degree at most 3. No stronger lower bound is known for the more general SNDP.

1.2. Our results. We show hardness of approximation for several of these vertex-connectivity network design problems. In section 2, we show that SNDP cannot be approximated within a ratio of $2^{\log^{1-\epsilon} n}$ for any fixed $\epsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\mathrm{polylog}(n)})$. This hardness of approximation result also extends to the subset k-connectivity problem which is a special case of SNDP. The lower bound holds for $k = n^{\rho}$, where $0 < \rho < 1$ is any fixed constant. It follows that when k/n is bounded away from 1, SNDP is provably harder to approximate than k-VCSS, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\mathrm{polylog}(n)})$.

In section 3, we show that the OSP cannot be approximated within a ratio of $(\frac{1}{2} - \epsilon) \ln n$ for any fixed $\epsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$. This hardness contrasts with other simple cases of SNDP. First, OSP with a general subset S is much harder to approximate than the special case $S = V \setminus \{r\}$ (which can be approximated within ratio 2). Second, this special case of SNDP is already much harder to approximate than the edge-connectivity variant of (general) SNDP (which can be approximated within ratio 2). Both claims assume, of course, that $\mathsf{NP} \not\subseteq \mathsf{DTIME}(n^{O(\log \log n)})$.

In section 4, we exhibit APX-hardness for $\mathrm{VCAP}_{1,2}$, even in the case where every pair of vertices in the graph forms an augmenting edge of cost 1 or 2. From this, it follows that $\mathrm{VCAP}_{k,k+1}$ with uniform costs is APX-hard for every $k \geq 2$. For fixed k, this hardness result matches, up to constant factors, the approximation algorithms mentioned in section 1.1.

Remark. SNDP with integer costs bounded by a polynomial in n can be reduced to SNDP with uniform costs. Indeed, one can replace every edge of cost $c > 0$ with a path consisting of c unit-cost edges, letting the new vertices have no connectivity requirement; i.e., $k_{u,v} = 0$ if $\{u, v\}$ contains a new vertex. Edges of cost 0 can be handled by changing their cost to, say, $1/n^3$, and then the reduction above is applicable (with a suitable scaling). It is straightforward that the argument above

regarding SNDP also holds for OSP and for the subset k-connectivity problem; thus our hardness of approximation results for these three problems hold even in the case of uniform costs.

1.3. Preliminaries. For an arbitrary graph G, let $V(G)$ denote the vertex set of G and let $E(G)$ denote the edge set of G. For a nonnegative cost function c on the edges of G and a subgraph $G' = (V', E')$ of G we use the notation $\text{cost}(G') = \text{cost}(E') = \sum_{e \in E'} c(e)$. We denote the set of *neighbors* of a vertex v in $W \subset V$ (namely, the vertices $w \in W$ such that $(v, w) \in E$) by $N(v, W, G)$. When $W = V(G)$ we omit W and write $N(v, G)$, and when G is clear from the context, we use simply $N(v)$.

A set W of k vertices in a graph $G = (V, E)$ is called a *k–vertex-cut* (or just a *vertex-cut*) if the subgraph of G induced on $V \setminus W$ is not connected. A vertex $w \in V$ is called a *cut-vertex* if $W = \{w\}$ is a vertex-cut. A graph is *k–vertex-connected* if there are k vertex-disjoint paths between every pair of vertices. We will use the following classical result.

THEOREM 1.1 (Menger's theorem; see, e.g., [8]).

(a) *A graph G contains at least k vertex-disjoint paths between two nonadjacent vertices u, v if and only if every vertex-cut that separates u from v must be of size at least k.*

(b) *A graph G is k–vertex-connected if and only if it has no $(k-1)$–vertex-cut.*

2. Survivable network design and subset connectivity. In this section, we exhibit a hardness result for approximating the subset connectivity problem, and thus also for SNDP, within a ratio of $2^{\log^{1-\epsilon} n}$ for any fixed $\epsilon > 0$. The lower bound is proved by a reduction from a graph-theoretic problem called MINREP, which is defined in [27]. This problem is closely related to the LABELCOVER$_{\max}$ problem of [1] and to the parallel repetition theorem of [33]. We first describe the MINREP problem and the hardness results known for it in section 2.1. We then give a reduction from MINREP to SNDP in section 2.2. Finally, we adapt this reduction to the subset connectivity problem in section 2.3.

2.1. The MINREP problem. Arora and Lund [1] introduced a problem called LABELCOVER$_{\max}$ as a graph-theoretic description of one-round two-prover proof systems for which the parallel repetition theorem of Raz [33] applies. The MINREP problem described below is closely related to LABELCOVER$_{\max}$ and was defined in [27] for the same purpose.

The input to the MINREP problem consists of a bipartite graph $G(A, B, E)$, with an explicit partitioning of each of A and B into equal-sized subsets, namely, $A = \bigcup_{i=1}^{q_A} A_i$ and $B = \bigcup_{j=1}^{q_B} B_j$, where all the sets A_i have the same size $m_A = |A|/q_A$ and all the sets B_j have the same size $m_B = |B|/q_B$. The bipartite graph G induces a *super-graph* H as follows. The *super-vertices* (i.e., the vertices of H) are the $q_A + q_B$ sets A_i and B_j. A *super-edge* (an edge in H) connects two super-vertices A_i and B_j if there exist some $a \in A_i$ and $b \in B_j$ which are adjacent in G.

A pair (a, b) *covers* a super-edge (A_i, B_j) if $a \in A_i$ and $b \in B_j$ are adjacent in G. Let $S \subseteq A_i \cup B_j$. (The vertices of S can be thought of as *representatives* from A_i and from B_j.) We say that S *covers* the super-edge (A_i, B_j) if there exist two vertices $a, b \in S$ such that the pair (a, b) covers the super-edge (A_i, B_j).

The goal in the MINREP problem is to select representatives from each set A_i and each set B_j such that all the super-edges are covered and the total number of representatives selected is minimal. That is, we wish to find subsets $A' \subseteq A$ and

$B' \subseteq B$ with minimal total size $|A'| + |B'|$, such that for every super-edge (A_i, B_j) there exist representatives $a \in A' \cap A_i$ and $b \in B' \cap B_j$ that are adjacent in G.

For our purposes, it is convenient (and possible) to restrict the MinRep problem so that for every super-edge (A_i, B_j), each vertex in B_j is adjacent to at most one vertex in A_i. We call this additional property of G the *star property* because it is equivalent to saying that for every super-edge (A_i, B_j) the subgraph of G induced on $A_i \cup B_j$ is a collection of vertex-disjoint stars whose centers are in A_i.[1] See Figure 1 for an illustration.

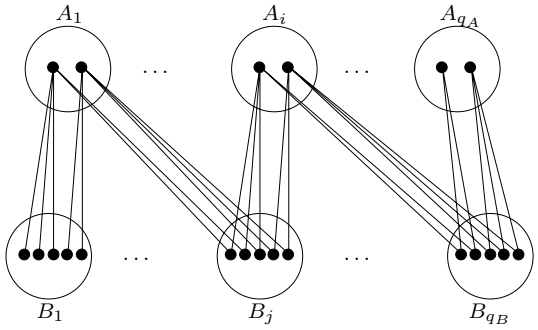


FIG. 1. *An instance of* MinRep *with the star property.*

The next theorem follows by a straightforward application of the parallel repetition theorem of Raz [33], since the MinRep problem is a graph-theoretic description of two-prover one-round proof systems. The additional star property is achieved by using a specific proof system. A description can be found in [12, section 2.2].

THEOREM 2.1. *Let $L \in \mathsf{NP}$ and fix $\epsilon > 0$. Then there exists an algorithm (a reduction), whose running time is quasi-polynomial, namely, $n^{\mathrm{polylog}(n)}$, and that given an instance x of L, produces an instance $G(A, B, E)$ of the* MinRep *problem with the star property, such that the following holds.*

- *If $x \in L$, then the* MinRep *instance G has a solution of value $q_A + q_B$ (namely, with one representative from each A_i and one from each B_j).*
- *If $x \notin L$, then the value of any solution of the* MinRep *instance G is at least $(q_A + q_B) \cdot 2^{\log^{1-\epsilon}|V(G)|}$.*

Hence, MinRep *cannot be approximated within ratio $2^{\log^{1-\epsilon} n}$, for any fixed $\epsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\mathrm{polylog}(n)})$.*

2.2. Hardness of survivable network design.

THEOREM 2.2. SNDP *cannot be approximated within ratio $2^{\log^{1-\epsilon} n}$, for any fixed $\epsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\mathrm{polylog}(n)})$.*

The reduction. The proof of Theorem 2.2 is by a reduction whose starting point is Theorem 2.1. Specifically, given the instance $G(A, B, E)$ of the MinRep problem as described in section 2.1, create an instance $\bar{G}(\bar{V}, \bar{E})$ of SNDP as follows. (See Figure 2 for an illustration.)

1. Take G and let all its edges have cost 0.
2. For each $i = 1, \ldots, q_A$ create a new vertex u_i that is connected to every vertex in A_i by an edge of cost 1. Similarly, for each $j = 1, \ldots, q_B$ create a

[1] A *star* is a graph all of whose vertices have degree 1, except for one vertex that may have degree larger than 1. This vertex is called the *center* of the star, and the other vertices are called *leaves* of the star.

new vertex w_j that is connected to every vertex in B_j by an edge of cost 1. (Informally, these edges correspond to choosing representatives from A_i and B_j.) Let $U = \{u_1, \ldots, u_{q_A}\}$ and $W = \{w_1, \ldots, w_{q_B}\}$.

3. For every super-edge (A_i, B_j) create two new vertices x_i^j and y_j^i. For every i, let $X_i = \{x_i^j : (A_i, B_j) \text{ is a super-edge}\}$ and connect every vertex of X_i to u_i by edges of cost 0. Similarly, for every j, let $Y_j = \{y_j{}^i : (A_i, B_j) \text{ is a super-edge}\}$ and connect every vertex in Y_j to w_j by an edge of cost 0. (Informally, the connectivity requirement between x_i^j and y_j^i "guarantees" that the super-edge (A_i, B_j) is covered.)
4. For every super-edge (A_i, B_j) connect every vertex in $\{x_i^j, y_j^i\}$ to every vertex in $(A \setminus A_i) \cup (B \setminus B_j)$ by an edge of cost 0.
5. Let $X = \cup_{i=1}^{q_A} X_i$ and $Y = \cup_{j=1}^{q_B} Y_j$. Connect every two vertices in $X \cup Y$ by an edge of cost 0.
6. Finally, require $k = |X| + |Y| + (q_A - 1)m_A + (q_B - 1)m_B$ vertex-disjoint paths from x_i^j to y_j^i for every super-edge (A_i, B_j).

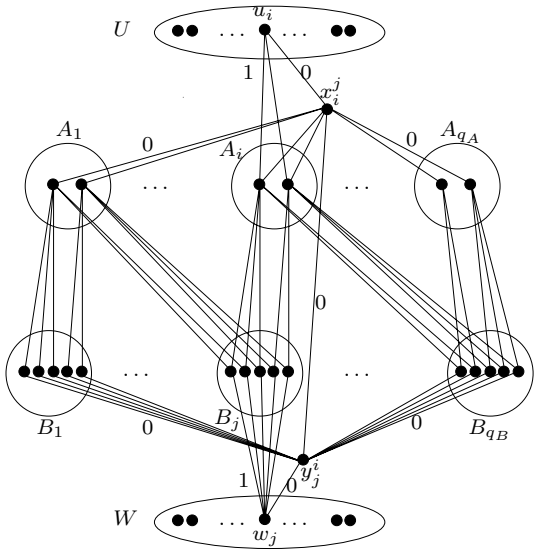


FIG. 2. *The vertices x_i^j, y_j^i in the* SNDP *instance $\bar{G}$.*

The analysis. Suppose $x \in L$ and then by Theorem 2.1 there exists a choice of $q_A + q_B$ representatives (one representative from each A_i and one from each B_j) that cover all the super-edges. Let G' be the subgraph of $\bar{G}$ that contains an edge between each u_i and the representative chosen in A_i, an edge between each w_j and the representative chosen in B_j, and all the edges of cost 0 in $\bar{G}$. Clearly, $\text{cost}(G') = q_A + q_B$. Let us now show that G' is a solution to the instance $\bar{G}$ of the SNDP. Consider a pair of vertices x_i^j, y_j^i such that (A_i, B_j) is a super-edge in G. Each vertex in $\mathcal{F}_{i,j} = (X \setminus \{x_i^j\}) \cup (Y \setminus \{y_j^i\}) \cup (A \setminus A_i) \cup (B \setminus B_j)$ defines a path of length 2 in G' between x_i^j and y_j^i, and the edge (x_i^j, y_j^i) defines a path of length 1, so we get a total of $|X| - 1 + |Y| - 1 + (q_A - 1)m_A + (q_B - 1)m_B + 1 = k - 1$ vertex-disjoint paths between x_i^j and y_j^i. There is an additional path that goes through $V \setminus \mathcal{F}_{i,j} = U \cup W \cup A_i \cup B_j \cup \{x_i^j, y_j^i\}$, namely, $x_i^j - u_i - a_i - b_j - w_j - y_j^i$, where

a_i and b_j are the representatives chosen from A_i and B_j, respectively. This path is clearly vertex-disjoint from the other $k-1$ paths, yielding a total of k vertex-disjoint paths between x_i^j and y_j^i.

The next lemma will be used to complete the proof of Theorem 2.2. Let G' be a feasible solution to the instance $\bar{G}$ of SNDP, i.e., a subgraph of $\bar{G}$ in which for every super-edge (A_i, B_j) there are k vertex-disjoint paths between x_i^j and y_j^i.

LEMMA 2.3. *For every super-edge (A_i, B_j), the subgraph G' contains an edge connecting u_i to some $a_i \in A_i$, and an edge connecting w_j to some $b_j \in B_j$, such that $(a_i, b_j) \in E$ (i.e., the pair (a_i, b_j) covers the super-edge).*

Proof. Since G' is a feasible solution, it contains k vertex-disjoint paths between x_i^j and y_j^i. Let $\mathcal{F}_{i,j} = (X \setminus \{x_i^j\}) \cup (Y \setminus \{y_j^i\}) \cup (A \setminus A_i) \cup (B \setminus B_j)$. At most $|\mathcal{F}_{i,j}| = |X|-1+|Y|-1+(q_A-1)m_A+(q_B-1)m_B = k-2$ of these k paths can visit vertices of $\mathcal{F}_{i,j}$, and at most one of these paths can use the edge (x_i^j, y_j^i). Hence, G' contains a path between x_i^j and y_j^i that visits only vertices of $\bar{V} \setminus \mathcal{F}_{i,j} = U \cup W \cup A_i \cup B_j \cup \{x_i^j, y_j^i\}$ and whose length is at least 2.

Observe that in the subgraph of G' induced on $\bar{V} \setminus \mathcal{F}_{i,j}$ the following holds. (Assume without loss of generality that G' contains all the edges of cost 0 in $\bar{G}$.) The only neighbor of x_i^j is u_i, so the vertices at distance 2 from x_i^j (i.e., the neighbors of u_i except for x_i^j) form a subset A_i' of A_i. Thus, the vertices at distance 3 from x_i^j (i.e., all the neighbors of A_i' except for u_i) are all from B_j. Similarly, the only neighbor of y_j^i is w_j, so vertices at distance 2 from y_j^i form a subset B_j' of B_j, and all vertices at distance 3 from y_j^i are from A_i. Note that the subgraph of $\bar{G}$ induced on $A_i \cup B_j$ is a collection of vertex-disjoint stars, whose centers are in A_i and whose leaves are in B_j. The aforementioned path in G' between x_i^j and y_j^i (that visits only vertices of $\bar{V} \setminus \mathcal{F}_{i,j}$) must then be of the form $x_i^j - u_i - a_i - b_j - w_j - y_j^i$ with $a_i \in A_i'$ and $b_j \in B_j'$ (note that the other vertices of $U \cup W$ are unreachable from x_i^j and y_j^i), and Lemma 2.3 follows. ∎

We now complete the proof of Theorem 2.2. Suppose that $x \notin L$ and let G' be a feasible solution to the instance $\bar{G}$ of the SNDP. Let A_i' be the set of neighbors of u_i among A_i (in G'), and let B_j' be the set of neighbors of w_j among B_j (in G'). By Lemma 2.3 the representatives $A' = \cup_i A_i'$ and $B' = \cup_j B_j'$ cover all the super-edges (A_i, B_j), thus forming a feasible solution to the MINREP instance G. By Theorem 2.1 the value of this MINREP solution, which is $|A'| + |B'|$, is at least $(q_A + q_B) \cdot 2^{\log^{1-\epsilon} n}$, where n denotes the number of vertices in G. Observe that $\text{cost}(G') = |A'| + |B'|$. Since $|V(\bar{G})| = |V(G)|^{O(1)}$, we get that $\text{cost}(G') \geq (q_A + q_B) \cdot 2^{\log^{1-\epsilon} |V(\bar{G})|}$, proving Theorem 2.2.

2.3. Hardness of subset *k*-connectivity. We can adapt the reduction of Theorem 2.2 to the subset k-connectivity problem as follows. We require that the subset $S = X \cup Y$ is k–vertex-connected. For this S to be k–vertex-connected in the case $x \in L$, we add, for every $z, z' \in S$ that are not a pair x_i^j, y_j^i, a set $Q_{z,z'}$ of k new vertices that are all connected to z and to z' by edges of cost 0. It can be seen that the analysis of the case $x \notin L$ (including the proof of Lemma 2.3) remains valid, and the number of vertices in the graph is still $|V(G)|^{O(1)}$. We thus obtain the following hardness of approximation result for the subset k-connectivity problem.

THEOREM 2.4. *The subset k-connectivity problem cannot be approximated within ratio $2^{\log^{1-\epsilon} n}$, for any fixed $\epsilon > 0$, unless* $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\text{polylog}(n)})$.

Note that in the reduction in Theorem 2.2 $|X| = |Y|$ is the number of super-edges in G, and thus $k = \Theta(|A \cup B| + |X|)$ while the number of vertices is $\Theta(|A \cup B| + k|X|^2)$.

Therefore, our hardness result for the subset k-connectivity applies for $k \geq \Omega(n^{1/3})$, where n denotes the number of vertices in the input graph. In this problem, it is straightforward to achieve $k = n^{\alpha}$ for any fixed $0 < \alpha < 1$ by adding sufficiently many vertices that are either isolated or connected to all other vertices by edges of cost 0. It follows that the min-cost subset k-connectivity problem is provably harder to approximate than the min-cost k-connectivity problem (for values of k as above). Indeed, it is shown in [5] that the latter problem can be approximated within ratio $O(\log k)$ for $k \leq \sqrt{n/6}$.

3. Outconnectivity to a subset. In this section we show a lower bound of $\Omega(\log n)$ for approximating the outconnectivity from a root to a subset problem (OSP).

THEOREM 3.1. *The OSP cannot be approximated within a ratio of* $(\frac{1}{2} - \epsilon) \ln n$ *for any fixed* $\epsilon > 0$, *unless* $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$.

For ease of exposition, we present the reduction to the OSP in stages by going through an intermediate problem which is easier to deal with. The 3-OSP is defined as OSP with the additional restriction that the k vertex-disjoint paths between r and each $s \in S$ are required to have length at most 3. Note that 3-OSP is not a special case of the SNDP. We give a hardness of approximation result for the 3-OSP in section 3.2 and for the OSP in section 3.3. The starting point for these reductions is a gap shown in [13] for the problem of packing set-covers, as described in section 3.1.

3.1. The set-cover packing problem. Let $G(V_1, V_2, E)$ be a bipartite graph. We say that a vertex $v_1 \in V_1$ *covers* a vertex $v_2 \in V_2$ if the two vertices are adjacent, i.e., $(v_1, v_2) \in E$. A *set-cover* (of V_2) in $G(V_1, V_2, E)$ is a subset $S \subseteq V_1$ such that every vertex of V_2 is covered by some vertex from S. Throughout, we assume that the intended bipartition (V_1, V_2) is given explicitly as part of the input, and that every vertex in V_2 can be covered (i.e., has at least one neighbor in V_1).

A *set-cover packing* in the bipartite graph G is a collection of pairwise-disjoint set-covers of V_2. The *set-cover packing problem* is to find in an input bipartite graph G (as above), a maximum number of pairwise-disjoint set-covers of V_2. We denote by $\mathrm{sc}^*(G)$ the minimum size of a set-cover of V_2 in G, and by $\mathrm{scp}^*(G)$ the maximum size of a set-cover packing of G. Note that $\mathrm{scp}^*(G) \leq |V_1|/\mathrm{sc}^*(G)$. Feige, Halldórsson, Kortsarz, and Srinivasan [13] give a hardness of approximation result for the set-cover packing problem by proving the following theorem.

THEOREM 3.2 (see [13]). *Let* $L \in \mathsf{NP}$ *and fix* $\epsilon > 0$. *Then there exists an algorithm (a reduction), whose running time is nearly polynomial, namely* $n^{O(\log \log n)}$, *and that given an instance* x *(for* L*), produces an instance* $G(V_1, V_2, E)$ *of the set-cover packing problem (and a number* $d \leq |V_1|$*) such that* $|V_1| \leq |V_2|^{\epsilon}$ *and the following holds.*

- *If* $x \in L$, *then* V_1 *can be partitioned into* d *equal-sized set-covers of* V_2. *(Therefore,* $\mathrm{scp}^*(G) \geq d$.*)*
- *If* $x \notin L$, *then the size of any set-cover of* V_2 *is at least* $(|V_1|/d) \cdot (1-\epsilon) \ln |V_1 \cup V_2|$. *(Therefore,* $\mathrm{scp}^*(G) \leq d/[(1-\epsilon) \ln |V_1 \cup V_2|]$.*)*

It is straightforward from this reduction that for any fixed $\epsilon > 0$, the set-cover packing problem cannot be approximated within ratio $(1 - \epsilon) \ln n$ (where n is the number of vertices in the graph), unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$.

3.2. Hardness of outconnectivity to a subset with path length 3. We prove the following theorem as an intermediate step towards proving hardness of approximation for the OSP.

THEOREM 3.3. 3-OSP *cannot be approximated within ratio* $(1-\epsilon)\ln n$*, for any fixed* $\epsilon > 0$*, unless* $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log\log n)})$.

The reduction. The proof of Theorem 3.3 is by a reduction whose starting point is Theorem 3.2. Specifically, given the set-cover packing instance $G(V_1, V_2, E)$ we construct from G a new graph $\bar{G}$ as follows. (See Figure 3 for illustration.) Add to G a set A of d new vertices (where d is the number from the reduction), and form a complete bipartite graph between A and V_1. Let all the edges of G have cost 0, and all the edges between A and V_1 have cost 1. Now add a new vertex r that will be the root, and connect it to each vertex of A by an edge of cost 0. Finally, set $S = V_2$ and $k = d$. That is, a feasible solution is a subgraph of $\bar{G}$ that contains at least d vertex-disjoint paths of length at most 3 between r and each $s \in S$.

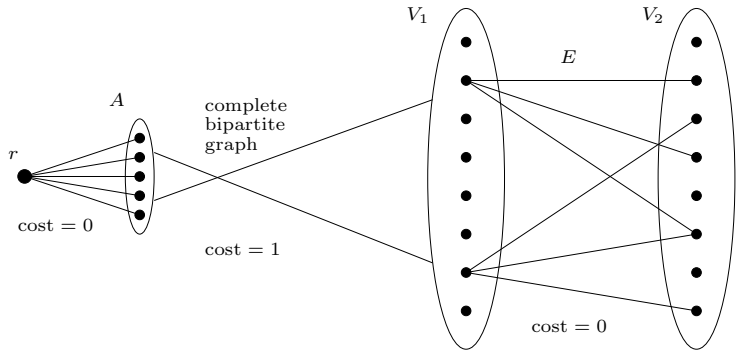


FIG. 3. *The graph* $\bar{G}$ *of the reduction from set-cover packing to* 3-OSP.

The analysis. Suppose $x \in L$ and then by Theorem 3.2 the set-cover packing graph $G(V_1, V_2, E)$ has a set-cover packing of size d. Let G' be a subgraph of $\bar{G}$ that contains all the edges of cost 0, and that connects each $a \in A$ to (all the vertices of) a set-cover N_a of V_2, such that the set-covers $\{N_a\}_{a\in A}$ are pairwise-disjoint. Such a subgraph G' exists since $|A| = d \le \mathrm{scp}^*(G)$. Since the edges of cost 1 in G' are incident at distinct vertices of V_1, we get that $\mathrm{cost}(G') \le |V_1|$.

To prove that G' is a feasible solution to 3-OSP we show d vertex-disjoint paths between r and any $v_2 \in V_2$. For every $a \in A$ we have that $N_a = N(a, V_1, G')$ is a set-cover of V_2 and thus contains a neighbor of v_2. Therefore, a defines a path of length 3 in G' between r and v_2. The $|A| = d$ paths that we obtain are vertex-disjoint because each vertex $a \in A$ is contained in exactly one of these paths, and because each vertex of V_1 belongs to at most one set-cover N_a.

The next lemma will be used to complete the proof of Theorem 3.3. Let G' be a feasible solution to the above described instance $\bar{G}$ of the 3-OSP problem.

LEMMA 3.4. *For every* $a \in A$*, the set* $N_a = N(a, V_1, G')$ *is a set-cover (in* G*) of* V_2 *(i.e., every* $a \in A$ *is at distance* 2 *in* G' *from every* $v_2 \in V_2$*).*

Proof. Let $a \in A$ and $v_2 \in V_2$; we will show that N_a covers v_2. Since $N(r, \bar{G}) = A$ and the distance in $\bar{G}$ between r and v_2 is 3, any path of length at most 3 between r and v_2 in G' must contain at least one vertex of A. Since G' is a feasible solution, it contains $d = |A|$ vertex-disjoint paths; these paths are vertex-disjoint and hence exactly one of these d paths must contain the vertex $a \in A$. In this path a is at distance 2 from v_2, implying that N_a covers v_2. □

We now complete the proof of Theorem 3.3. Suppose $x \notin L$. Let G' be a feasible solution to $\bar{G}$ and let $N_a = N(a, V_1, G')$, as in the above analysis of Theorem 3.3.

Clearly, $\text{cost}(G') = \sum_{a \in A} |N_a|$. By Lemma 3.4, each set N_a forms a set-cover (in G) of V_2. By Theorem 3.2 (and since $|A| = d$) we get that $\text{cost}(G') \geq |A| \cdot \text{sc}^*(G) \geq (1-\epsilon)|V_1| \cdot \ln |V_1 \cup V_2|$. Note that $|V(\bar{G})| = |V_1 \cup V_2| + d + 1 \leq 2|V_1 \cup V_2|$ and thus the gap between the case $x \in L$ and the case $x \notin L$ is at least $(1-\epsilon) \ln |V_1 \cup V_2| \geq (1-2\epsilon) \ln |V(\bar{G})|$, proving Theorem 3.3.

3.3. Hardness of outconnectivity to a subset. We now prove Theorem 3.1, i.e., that OSP cannot be approximated within ratio $(\frac{1}{2} - \epsilon) \ln n$, for any fixed $\epsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$. Observe that the reduction to 3-OSP (in section 3.2) might not work for OSP because in the case $x \notin L$, a feasible solution G' might connect r and $v_2 \in V_2$ by d long paths (namely, of length more than 3), where each path contains one (distinct) vertex of A. However, each of these paths must contain at least one vertex of $V_2 \setminus \{v_2\}$, so at most $|V_2|$ such paths are vertex-disjoint. Here we use the special properties of the set-cover packing problem; by duplicating V_1 sufficiently many times, we increase $\text{scp}^*(G)$ and, accordingly, the connectivity requirement k, so that they are both much larger than $|V_2|$, ensuring that paths of length more than 3 have only a negligible effect in any feasible solution.

The reduction. Define a *copy* of a vertex v in a graph as a new vertex v' that is connected by edges to the same vertices as v, and with the same edge costs. In the reduction below, we replace certain vertices by many copies of them. Let us denote by $\tilde{v}$ the set of all *copies* of v. Note that no two vertices in $\tilde{v}$ are connected by an edge. For a set of vertices $W = \{w_1, w_2, \ldots\}$, let $\tilde{W} = \bigcup_i \tilde{w}_i$ be the set of all copies of all vertices in W.

The proof of Theorem 3.3 is by a reduction whose starting point is Theorem 3.2. Specifically, given the set-cover packing instance $G(V_1, V_2, E)$ construct a new graph $\tilde{G}$ as follows. First, add to G a set $A = \{a_1, \ldots, a_d\}$ of d new vertices that are connected by a complete bipartite graph to V_1, letting all the edges of G have cost 0 and all the edges between A and V_1 have cost 1. Next, add a new vertex r that will be the root, and connect it to each vertex of A by an edge of cost 0. (So far, this graph is $\bar{G}$ from section 3.2.) Now, replace each vertex of $A \cup V_1$ by $|V_2|^2$ copies of it. Thus, $\tilde{A} = \bigcup_{i=1}^{d} \tilde{a}_i$, where $\tilde{a}_i$ is the set of $|V_2|^2$ copies of a_i, and $\tilde{V}_1 = \bigcup_{v \in V_1} \tilde{v}$, where $\tilde{v}$ is the set of $|V_2|^2$ copies of v. Finally, set $S = V_2$, $k = |V_2|^2 d$. That is, a feasible solution is a subgraph of $\tilde{G}$ that contains at least k vertex-disjoint paths between r and each $s \in S$.

The analysis. Throughout the proof, let *set-cover in* $\tilde{G}$ refer to a set-cover of V_2 by vertices of $\tilde{V}_1$ in the bipartite graph that $\tilde{G}$ induces on $\tilde{V}_1 \cup V_2$. Observe that the minimum size of a set-cover of V_2 in $\tilde{G}$ is the same as in G; i.e., $\text{sc}^*(\tilde{G}) = \text{sc}^*(G)$. Also, $\text{scp}^*(\tilde{G}) \geq |V_2|^2 \cdot \text{scp}^*(G)$ since a set-cover packing of G has $|V_2|^2$ pairwise-disjoint copies in $\tilde{G}$.

Suppose $x \in L$ and then by Theorem 3.2 the set-cover packing graph $G(V_1, V_2, E)$ has a set-cover packing of size d. It follows that $\text{scp}^*(\tilde{G}) \geq |V_2|^2 d$. Now an argument identical to the one in section 3.2 shows a subgraph G' that is a feasible solution to OSP and with $\text{cost}(G') \leq |\tilde{V}_1| = |V_2|^2 |V_1|$.

Lemmas 3.5 and 3.6 will be used to complete the proof of Theorem 3.1. They are essentially analogous to Lemma 3.4. Let G' be a feasible solution to the instance $\tilde{G}$ of the OSP.

LEMMA 3.5. *For every $v_2 \in V_2$, less than $|V_2|$ vertices of $\tilde{A}$ are not at distance* 2 *in G' from v_2.*

Proof. Since G' is a feasible solution to the OSP instance $\tilde{G}$, it must contain at least k vertex-disjoint paths between v_2 and r. The $k = |\tilde{A}|$ paths are disjoint but

they all have to go through $\tilde{A}$, and thus each vertex of $\tilde{A}$ must belong to exactly one of these paths. Now, if a vertex of $\tilde{A}$ is not at distance 2 from v_2, then the path containing it must visit at least one additional vertex of V_2. But since the paths are disjoint, this event happens less than $|V_2|$ times. □

LEMMA 3.6. *There exists a feasible solution* G'_+ *with* $\text{cost}(G'_+) \leq \text{cost}(G') + |V_2|^2$, *such that for every* $a \in \tilde{A}$ *the set* $N(a, \tilde{V}_1, G'_+)$ *is a set-cover (in* $\tilde{G}$*) of* V_2.

Proof. Augment the feasible solution G' to a graph G'_+ as follows. For every $v_2 \in V_2$ and every $a \in \tilde{A}$, if a is not at distance 2 in G' from v_2, then add to G' an edge between a and an arbitrary vertex in $N(v_2, \tilde{G})$. By Lemma 3.5, every $v_2 \in V_2$ causes the addition of at most $|V_2|$ edges. Since each added edge has cost 1, the resulting G'_+ is a feasible solution with $\text{cost}(G'_+) \leq \text{cost}(G') + |V_2|^2$. Furthermore, every $a \in \tilde{A}$ is at distance 2 in G'_+ from every vertex $v_2 \in V_2$; i.e., the set $N(a, \tilde{V}_1, G'_+)$ is a set-cover (in G) of V_2. □

We now complete the proof of Theorem 3.1. Suppose $x \notin L$ and let G' be a feasible solution to $\tilde{G}$, as above. By Theorem 3.2 we have $\text{sc}^*(\tilde{G}) = \text{sc}^*(G) \geq (|V_1|/d) \cdot (1-\epsilon) \ln |V_1 \cup V_2|$. Let G'_+ be the augmented solution that follows from Lemma 3.6. Then for every $a \in \tilde{A}$, the set $N(a, \tilde{V}_1, G'_+)$ is a set-cover (in $\tilde{G}$) of V_2. Therefore, $\text{cost}(G'_+) = \sum_{a \in \tilde{A}} |N(a, \tilde{V}_1, G'_+)| \geq |\tilde{A}| \cdot \text{sc}^*(\tilde{G}) \geq |V_2|^2 |V_1| \cdot (1-\epsilon) \ln |V_1 \cup V_2|$. It follows that $\text{cost}(G') \geq \text{cost}(G'_+) - |V_2|^2 \geq |V_2|^2 |V_1| \cdot (1-2\epsilon) \ln |V_1 \cup V_2|$.

Since $d \leq |V_1| \leq |V_2|^\epsilon$, we have that $|V(\tilde{G})| = |V_2| + (|V_1| + d) \cdot |V_2|^2 \leq 3|V_1| \cdot |V_2|^2 \leq |V_2|^{2+2\epsilon}$. Thus, the gap between the case $x \in L$ and the case $x \notin L$ is at least $(1-2\epsilon) \ln |V_1 \cup V_2| \geq \frac{1-2\epsilon}{2+2\epsilon} \ln |V(\tilde{G})| \geq (\frac{1}{2} - 2\epsilon) \ln |V(\tilde{G})|$, proving Theorem 3.1.

4. Vertex-connectivity augmentation. In this section we show APX-hardness for the following vertex-connectivity augmentation problem ($\text{VCAP}_{\ell,k}$): Given a k-connected graph $G_0 = (V, E_0)$ and a cost function $c : V \times V \to \mathbb{N}$, find a set $E_1 \subseteq V \times V$ of minimum cost so that $G_1 = (V, E_0 \cup E_1)$ is ℓ-connected. Since all graphs considered here are simple, we will not allow G_1 to contain self-loops. $\text{VCAP}_{k,\ell}(a, b)$ will represent a version of the problem where edges have only cost a or b (so that $c : V \times V \to \{a, b\}$). The main result of this section is that for some fixed $\epsilon > 0$ and for every $k \geq 1$, it is NP-hard to approximate $\text{VCAP}_{k,k+1}(1, 2)$ within a factor of $1 + \epsilon$; this holds even in the case of uniform costs, i.e., $\text{VCAP}_{k,k+1}(1, \infty)$.

It is possible to convert any instance of $\text{VCAP}_{k_0,k_0+\alpha}$ to an "equivalent" instance of $\text{VCAP}_{k_0+1,k_0+1+\alpha}$ by adding to G_0 a new vertex that is connected to every old vertex. In addition, it will be immediate that our proof extends to edge costs from the set $\{1, \infty\}$. It thus suffices to prove the following.

THEOREM 4.1. *For any* $k \geq 1$ *and some fixed* $\epsilon > 0$ *(independent of* k*), it is* NP*-hard to approximate* $\text{VCAP}_{1,2}(1, 2)$ *within a factor of* $1 + \epsilon$.

The proof of Theorem 4.1 employs a reduction from 3-dimensional matching (3DM) that was used in [17] to prove that solving $\text{VCAP}_{1,2}(1, 2)$ (optimally) is NP-hard. We obtain a stronger result (hardness of approximation) by a more involved analysis of the reduction and by relying on the hardness of approximating a bounded version of the 3DM problem shown in [30].

3-*dimensional matching* (3DM) is the following problem. Given three (disjoint) sets W, X, Y, with $|W| = |Y| = |Z|$, and a set of hyperedges $M \subseteq W \times Y \times Z$, find the largest subset $M' \subseteq M$ which is a *matching*; i.e., if $(w, x, y), (w', x', y') \in M'$, then $w \neq w'$, $x \neq x'$, and $y \neq y'$. For any $z \in W \cup X \cup Y$, let $\deg(z)$ be the number of hyperedges in M that contain z. We define the *maximum degree* of an instance to be $\Delta = \max_{z \in W \cup X \cup Y} \deg(z)$. For an instance $\mathcal{I}$ of 3DM, let $\text{3DM}(\mathcal{I})$ be the size of an

optimal matching. For an instance $\mathcal{J}$ of $\text{VCAP}_{1,2}(1,2)$, let $\text{VCAP}(\mathcal{J})$ be the cost of an optimal augmentation.

The reduction. Let $\mathcal{I} = (M, W, X, Y)$ be an instance of 3DM with $|M| = p$ and $|W| = |X| = |Y| = q$. We create an instance $\mathcal{J}$ of $\text{VCAP}_{1,2}$ as follows. (See Figure 4 for illustration.) Let $G_0 = (V, E_0)$ with

$$V = \{r, \bar{r}\} \cup \{w_i, \bar{w}_i, x_i, y_i : i = 1, 2, \ldots, q\} \cup \{a_{ijk}, \bar{a}_{ijk} : (w_i, x_j, y_k) \in M\},$$

$$E_0 = \{(r, \bar{r})\} \cup \{(w_i, \bar{w}_i), (w_i, r), (x_i, \bar{r}), (y_i, r) : i = 1 \ldots, q\}$$
$$\cup \{(a_{ijk}, w_i), (\bar{w}_i, \bar{a}_{ijk}) : (w_i, x_j, y_k) \in M\}.$$

We will define $\text{cost}(\bar{a}_{ijk}, a_{ijk}) = \text{cost}(x_j, \bar{a}_{ijk}) = \text{cost}(y_k, a_{ijk}) = 1$ if $(w_i, x_j, y_k) \in M$ and $\text{cost}(u, v) = 2$ for all other $(u, v) \in V \times V$.

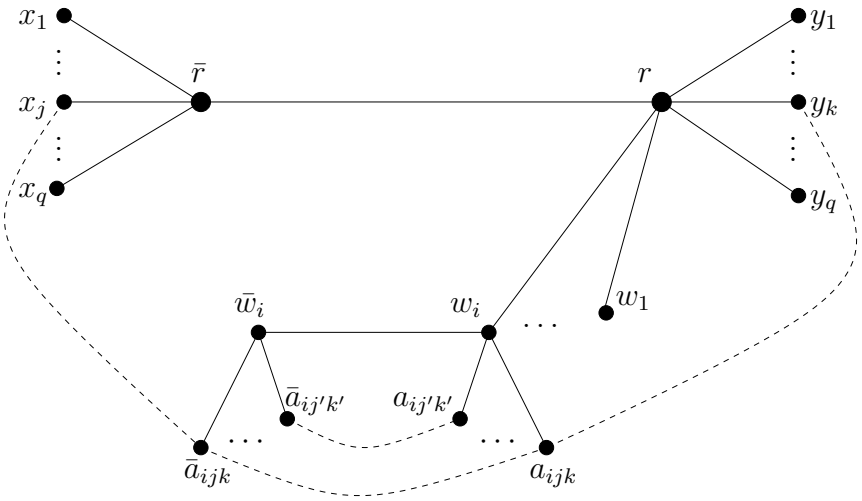


FIG. 4. *An instance of* VCAP *produced by the reduction. The solid lines represent edges of* G_0. *Some cost* 1 *edges are represented by dashed lines.*

LEMMA 4.2. *If* $\text{3DM}(\mathcal{I}) = q$, *then* $\text{VCAP}(\mathcal{J}) = p + q$.

Proof. Letting $M' \subseteq M$ be a matching of size q, we will construct an augmenting set E_1 consisting of $p + q$ edges of cost 1. These edges will be $(x_j, \bar{a}_{ijk})$ and (y_k, a_{ijk}) for every $(w_i, x_j, y_k) \in M'$ and $(a_{ijk}, \bar{a}_{ijk})$ for $(w_i, x_j, y_k) \in M - M'$. We must show that $G_1 = (V, E_0 \cup E_1)$ is 2-connected. By Menger's theorem (see section 1.3) it suffices to show that G_1 contains no cut-vertex.

Notice that G_0 is a tree with the $2(p+q)$ leaves $X \cup Y \cup \{a_{ijk}, \bar{a}_{ijk} : (w_i, x_j, y_k) \in M\}$. Neither of these leaves is a cut-vertex in G_0, and hence the same is true in G_1. So it remains to verify that each of r, $\bar{r}$, w_i, and $\bar{w}_i$ also is not a cut-vertex in G_1. It is easy to see that this indeed holds; for instance, if we remove some $\bar{w}_i$ from the graph, we may risk cutting off the vertices $\{\bar{a}_{ijk}\}$, but there is always some edge, either to a_{ijk} or to x_j (depending on whether $(w_i, x_j, y_k) \in M'$ or not), which leads back to the rest of the graph. Similar arguments hold for $r, \bar{r}$, and w_i. It follows that G_1 is 2-connected.

Hence E_1 is an augmenting set of cost $p+q$. To see that this is the cheapest such set, notice that for G_1 to be 2-connected, each leaf of G_0 must be incident to at least

one edge from E_1. Since there are $2(p+q)$ leaves and a single edge is incident to at most two of them, it follows that at least $p+q$ edges are necessary. ∎

LEMMA 4.3. *If* $\mathrm{VCAP}(\mathcal{J}) \le (p+q)(1+\epsilon)$, *then* $3\mathrm{DM}(\mathcal{I}) \ge q-(2+10\Delta)(p+q)\epsilon$.

Proof. Let $E_1 \subseteq E$ be a set of augmenting edges of cost at most $(p+q)(1+\epsilon)$ such that $G_1 = (V, E_0 \cup E_1)$ is 2-connected. As in Lemma 4.2, G_0 is a tree with the $2(p+q)$ leaves $X \cup Y \cup \{a_{ijk}, \bar{a}_{ijk} : (w_i, x_j, y_k) \in M\}$. Each leaf of G_0 must be adjacent to at least one edge of E_1 for G_1 to be 2-connected. Call a leaf *permissible* if it is adjacent to exactly one edge of E_1 and that edge has cost 1. Call a leaf *impermissible* otherwise (i.e., it is incident upon at least one edge of E_1 of cost 2 or upon more than one edge of E_1).

We first claim that at most $2(p+q)\epsilon$ leaves are impermissible. Indeed, for every impermissible leaf, the total cost of edges of E_1 that are incident at this leaf is at least 2. Similarly, for every permissible leaf this cost is exactly 1. The sum of these costs over all leaves is at most $2 \cdot \mathrm{cost}(E_1)$, since the cost of every edge of E_1 is counted at most twice (once from every end). Thus,

$$\#\{\text{permissible leaves}\} + 2 \cdot \#\{\text{impermissible leaves}\} \le 2 \cdot \mathrm{cost}(E_1) \le 2(p+q)(1+\epsilon).$$

Observing that the left-hand side is just $2(p+q)+\#\{\text{impermissible leaves}\}$, the claim follows immediately.

We now construct a set M' which is almost a matching. Initially, let $M' = \emptyset$. Then iteratively for $j = 1, 2, \ldots, q$ we try to find a hyperedge (in M) that contains x_j and add it to M', as follows. If x_j is permissible, then it is adjacent to a cost 1 edge of E_1; hence it is adjacent to some leaf $\bar{a}_{ijk}$. If both $\bar{a}_{ijk}$ and a_{ijk} are permissible, then the latter is adjacent (via a cost 1 edge) to some leaf y_k. If this leaf y_k is permissible, then add the hyperedge (w_i, x_j, y_k) to M'. Notice that $M' \subseteq M$ since the above process relies on cost 1 edges.

We next claim that $|M'| \ge q - 2\Delta(p+q)\epsilon$. Indeed, an impermissible x_j, a_{ijk}, or $\bar{a}_{ijk}$ can cause only one iteration (namely, the one with the corresponding value of j) to fail. An impermissible y_k can cause at most Δ iterations to fail, since it can be connected by edges of cost 1 to at most Δ leaves a_{ijk}. Denoting the number of impermissible y_k by n_y, we have that the number of iterations that fail is at most $2(p+q)\epsilon - n_y + n_y\Delta$. Since our claim shows that $n_y \le 2(p+q)\epsilon$, this is at most $2\Delta(p+q)\epsilon$.

By our construction, M' is almost a matching; its hyperedges have distinct elements from X and from Y, but its elements from W might be repeated, i.e., not distinct. For every element w_i that belongs to more than one hyperedge in M', let us remove from M' all but one of the hyperedges that contain w_i. The resulting set of hyperedges, denoted M'', is thus a matching. Let $\mu = q - |M''|$ be the number of vertices w_i that do not appear in any hyperedge of M' (or equivalently, of M''). Notice that $|M'| - |M''| \le q - |M''| = \mu$, so an upper bound on μ yields a lower bound on the size of the matching M''.

We now show that $\mu \le (2+8\Delta)(p+q)\epsilon$. Let E_1' be the edges of E_1 that correspond to hyperedges in M', namely, those edges $\{(x_j, \bar{a}_{ijk}) \text{ and } (y_k, a_{ijk})\}$ for $(w_i, x_j, y_k) \in M'$. We have that $\mathrm{cost}(E_1') \ge 2|M'| \ge 2q - 4\Delta(p+q)\epsilon$; hence

$$\mathrm{cost}(E_1 \setminus E_1') \le (p+q)(1+\epsilon) - 2q + 4\Delta(p+q)\epsilon = p - q + (1+4\Delta)(p+q)\epsilon. \quad (4.1)$$

Recall that each leaf (of G_0) a_{ijk} or $\bar{a}_{ijk}$ must be incident to an edge of E_1. The edges of E_1' are incident, by their definition, to at most $2|M'| \le 2q$ distinct such leaves; thus, the edges of $E_1 \setminus E_1'$ must be incident to the (at least) $2p - 2q$ remaining leaves

a_{ijk} and $\bar{a}_{ijk}$. If we split the cost of every edge in $E_1 \setminus E_1'$ (evenly) between its two endpoints, then we get that at least $2p - 2q$ leaves are each charged a cost of at least $1/2$. It follows that

$$\text{cost}(E_1 \setminus E_1') \geq (2p - 2q) \cdot (1/2). \tag{4.2}$$

We shall now improve over the lower bound (4.2) by considering the μ vertices w_i which do not make an appearance in M'. Each such w_i is a cut-vertex of $(V, E_0 \cup E_1')$ (by definition of E_1'), since its removal disconnects $W_i = \{\bar{w}_i\} \cup \{a_{ijk}, \bar{a}_{ijk} : (w_i, x_j, y_k) \in M\}$ from the rest of the graph. But w_i cannot be a cut-vertex of G_1, and thus $E_1 \setminus E_1'$ must contain an edge that connects W_i to the rest of the graph. We have three cases for this edge: (i) if it is incident (in W_i) to $\bar{w}_i$, then the edge's cost is at least 2 and $\bar{w}_i$ is charged at least 1; (ii) if the edge is incident (in W_i) to some a_{ijk} or $\bar{a}_{ijk}$ and (in the rest of the graph) to some $a_{i'j'k'}$ or $\bar{a}_{i'j'k'}$ (with $i \neq i'$), then the edge's cost is 2, and the endpoint in W_i is actually charged $1/2$ more than in the lower bound (4.2); or (iii) if this edge is incident (in W_i) to some a_{ijk} or $\bar{a}_{ijk}$ and (in the rest of the graph) to a vertex that is not $a_{i'j'k'}$ or $\bar{a}_{i'j'k'}$, then the edge's cost is at least 1, so the endpoint not in W_i is charged at least $1/2$. In all three cases, the fact that w_i is a cut-vertex in $(V, E_0 \cup E_1')$ implies that the lower bound (4.2) can be increased by $1/2$. It is easy to see that the increases corresponding to different w_i's are distinct, and thus,

$$\text{cost}(E_1 \setminus E_1') \geq (2p - 2q) \cdot (1/2) + \mu \cdot (1/2). \tag{4.3}$$

Combining equations (4.1) and (4.3) we indeed get that $\mu \leq (2 + 8\Delta)(p + q)\epsilon$. We conclude that $\mathcal{I}$ contains a matching M'' of size

$$|M''| \geq |M'| - \mu \geq q - 2\Delta(p + q)\epsilon - (2 + 8\Delta)(p + q)\epsilon = q - (2 + 10\Delta)(p + q)\epsilon,$$

which completes the proof of Lemma 4.3. □

3DM-5 is a bounded version of the 3DM problem in which every element of $W \cup Y \cup Z$ can appear at most five times in a triple of M, i.e., one in which $\Delta = 5$. It is shown in [30] that this variant is Max SNP-hard. In particular, the following theorem is proved.

THEOREM 4.4 (Petrank [30]). *For some fixed $\epsilon_0 > 0$, it is* NP*-hard to distinguish whether an instance of* 3DM-5 *with $|W| = |X| = |Y| = q$ has a perfect matching (of size q) or every matching has size at most $(1 - \epsilon_0)q$.*

If $|M| = p$ and $|W| = |X| = |Y| = q$, then in any instance of 3DM-5, we must have $p \leq 5q$. This observation, together with Lemmas 4.2 and 4.3 and Theorem 4.4, yield a proof of Theorem 4.1.

Proof of Theorem 4.1. We will show that our reduction above (see Theorem 4.1) is gap-preserving. Specifically, we will show that if $\mathcal{I}$ is an instance of 3DM-5 and $\mathcal{J}$ is the corresponding instance of $\text{VCAP}_{1,2}(1, 2)$, then

$$\begin{aligned} \text{3DM}(\mathcal{I}) = q &\Longrightarrow \text{VCAP}(\mathcal{J}) = p + q, \\ \text{3DM}(\mathcal{I}) < q(1 - \epsilon_0) &\Longrightarrow \text{VCAP}(\mathcal{J}) > (p + q)\,(1 + \epsilon_0/312)\,. \end{aligned}$$

The first implication follows directly from Lemma 4.2. The second one is the contrapositive of Lemma 4.3 with $\epsilon = \frac{\epsilon_0}{312}$, and then $\text{3DM}(\mathcal{I}) \geq q - (2 + 10\Delta)(5q + q)\epsilon = q(1 - 312\epsilon) = q(1 - \epsilon_0)$. □

Remark. A similar analysis can be applied to the NP-hardness reduction of [17] for the edge-connectivity augmentation problem (ECAP). This would prove that for

any $k \geq 1$ and some fixed $\epsilon > 0$ (independent of k), it is NP-hard to approximate $\mathrm{ECAP}_{k,k+1}(1,2)$ within a factor of $1+\epsilon$. The same holds for a model with uniform edge costs.

5. Discussion. We have shown that, in terms of approximation, the vertex-connectivity variant of SNDP differs significantly from the edge-connectivity variant, and that this holds even in relatively simple special cases. But there are a few important special cases which remain open. Most notably, for k-VCSS there is still a large gap between the known upper and lower bounds. It is particularly interesting that no result is known to exclude a 2-approximation; such a result would separate this problem from its edge-connectivity counterpart. The techniques that we relied on in section 3 were successfully applied to various problems to achieve (roughly) logarithmic hardness of approximation. It was our hope that these powerful techniques might also be applied to k-VCSS, but we were not able to do so.

An important observation to keep in mind is that the approximation ratios of SNDP and of k-VCSS are nondecreasing with the maximum requirement $k_{\max} := \max\{k_{u,v} : u, v \in V\}$. Indeed, given an instance graph with n vertices and maximum requirement $k_{\max}$, one can add a new vertex that is connected to all the existing vertices with zero-cost edges and increase all the existing requirements by 1. It is easy to see that any feasible solution to the original instance corresponds to a feasible solution with the same cost in the new instance, while $k_{\max}$ is increased by 1. It follows that any approximation ratio $f(k)$ (that is independent of n) must be nondecreasing with k. This argument also extends to the uniform cost case of SNDP by the remark at the end of section 1.2.

This observation may underlie two perplexing aspects of k-VCSS: (i) The known approximation ratio significantly degrades (approaches $\sqrt{n}$) as k gets closer to n, and one may suspect that this is not a coincidence. (ii) An interesting open question is whether the asymptotic approximation ratio of uniform cost k-VCSS is $1 + \Theta(1/k)$. Such an approximability threshold is known to exist for the Max k-Cut problem [23]. Nevertheless, general cost k-VCSS has completely different asymptotics; the result of [7] in conjunction with the observation in the previous paragraph shows that there is a fixed $\epsilon > 0$, such that for all $k \geq 2$, it is NP-hard to $1+\epsilon$ approximate k-VCSS with edge costs 0 and 1.

Finally, we stress that our reduction in section 2.2 relies on what we call the star property (in our graph-theoretic description) and which some literature refers to as the projection test. The hardness result of [10] improves over Theorem 2.1 by achieving a slightly larger inapproximability factor and by assuming the weaker complexity assumption $\mathsf{P} \neq \mathsf{NP}$. However, it lacks the star property that we require, and thus cannot be used to strengthen our result for SNDP.

Acknowledgments. We thank Joseph Cheriyan for raising the question of SNDP with uniform costs, which led to the remark at the end of section 1.2. We are also grateful to David Williamson and to the anonymous referees for comments that improved the presentation and for pointing out some of the issues mentioned in section 5.

REFERENCES

[1] S. Arora and C. Lund, *Hardness of approximations*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing, Boston, 1996.

[2] V. Auletta, Y. Dinitz, Z. Nutov, and D. Parente, *A 2-approximation algorithm for finding an optimum 3-vertex-connected spanning subgraph*, J. Algorithms, 32 (1999), pp. 21–30.

[3] J. Cheriyan, T. Jordán, and Z. Nutov, *On rooted node-connectivity problems*, Algorithmica, 30 (2001), pp. 353–375.
[4] J. Cheriyan and R. Thurimella, *Approximating minimum-size k-connected spanning subgraphs via matching*, SIAM J. Comput., 30 (2000), pp. 528–560.
[5] J. Cheriyan, S. Vempala, and A. Vetta, *Approximation algorithms for minimum-cost k-vertex connected subgraphs*, in 34th Annual ACM Symposium on Theory of Computing, Montreal, Quebec, Canada, 2002, pp. 306–312.
[6] B. Csaba, M. Karpinski, and P. Krysta, *Approximability of dense and sparse instances of minimum 2-connectivity, TSP and path problems*, in Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2002, pp. 74–83.
[7] A. Czumaj and A. Lingas, *On approximability of the minimum-cost k-connected spanning subgraph problem*, in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 281–290.
[8] R. Diestel, *Graph Theory*, 2nd ed., Springer-Verlag, New York, 2000.
[9] Y. Dinitz and Z. Nutov, *A 3-approximation algorithm for finding optimum* $4,5$*-vertex-connected spanning subgraphs*, J. Algorithms, 32 (1999), pp. 31–40.
[10] I. Dinur and S. Safra, *On the hardness of approximating label cover*, Technical report TR99-015, Electronic Colloquium on Computational Complexity, 1999.
[11] K. P. Eswaran and R. E. Tarjan, *Augmentation problems*, SIAM J. Comput., 5 (1976), pp. 653–665.
[12] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.
[13] U. Feige, M. M. Halldórsson, G. Kortsarz, and A. Srinivasan, *Approximating the domatic number*, SIAM J. Comput., 32 (2002), pp. 172–195.
[14] L. Fleischer, K. Jain, and D. P. Williamson, *An iterative rounding 2-approximation algorithm for the element connectivity problem*, in 42nd Annual IEEE Symposium on Foundations of Computer Science, Los Alamitos, CA, 2001, IEEE Computer Society, pp. 339–347.
[15] A. Frank, *Connectivity augmentation problems in network design*, in Mathematical Programming: State of the Art 1994, J. R. Birge and K. G. Murty, eds., The University of Michigan, Ann Arbor, MI, 1994, pp. 34–63.
[16] A. Frank and É. Tardos, *An application of submodular flows*, Linear Algebra Appl., 114/115 (1989), pp. 329–348.
[17] G. N. Frederickson and J. Ja'Ja', *Approximation algorithms for several graph augmentation problems*, SIAM J. Comput., 10 (1981), pp. 270–283.
[18] T. Hsu, *On four-connecting a triconnected graph*, J. Algorithms, 35 (2000), pp. 202–234.
[19] T. Hsu and V. Ramachandran, *A linear time algorithm for triconnectivity augmentation*, in 32nd Annual IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1991, IEEE Computer Society, pp. 548–559.
[20] K. Jain, *A factor 2 approximation algorithm for the generalized Steiner network problem*, Combinatorica, 21 (2001), pp. 39–60.
[21] K. Jain, I. Măndoiu, V. V. Vazirani, and D. P. Williamson, *A primal-dual schema based approximation algorithm for the element connectivity problem*, in Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 484–489.
[22] T. Jordán, *On the optimal vertex-connectivity augmentation*, J. Combin. Theory Ser. B, 63 (1995), pp. 8–20.
[23] V. Kann, S. Khanna, J. Lagergren, and A. Panconesi, *On the hardness of approximating* max k-cut *and its dual*, Chicago J. Theoret. Comput. Sci., 1997 (1997), Article 2.
[24] S. Khuller, *Approximation algorithms for finding highly connected subgraphs*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing, Boston, 1996.
[25] S. Khuller and B. Raghavachari, *Improved approximation algorithms for uniform connectivity problems*, J. Algorithms, 21 (1996), pp. 434–450.
[26] S. Khuller and R. Thurimella, *Approximation algorithms for graph augmentation*, J. Algorithms, 14 (1993), pp. 214–225.
[27] G. Kortsarz, *On the hardness of approximating spanners*, Algorithmica, 30 (2001), pp. 432–450.
[28] G. Kortsarz and Z. Nutov, *Approximating node connectivity problems via set covers*, in Approximation Algorithms for Combinatorial Optimization, Lecture Notes in Comput. Sci. 1913, Springer-Verlag, Berlin, 2000, pp. 194–205.
[29] G. Kortsarz and Z. Nutov, *Approximating k-node connected subgraphs via critical graphs*, in Proceedings of the 36th ACM Symposium on Theory of Computing, 2004, to appear.

[30] E. Petrank, *The hardness of approximation: Gap location*, Comput. Complexity, 4 (1994), pp. 133–157.
[31] R. Ravi and D. P. Williamson, *An approximation algorithm for minimum-cost vertex-connectivity problems*, Algorithmica, 18 (1997), pp. 21–43.
[32] R. Ravi and D. P. Williamson, *Erratum: An approximation algorithm for minimum-cost vertex-connectivity problems*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2002, pp. 1000–1001.
[33] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.
[34] V. V. Vazirani, *Approximation Algorithms*, Springer-Verlag, Berlin, 2001.
[35] T. Watanabe and A. Nakamura, *A minimum 3-connectivity augmentation of a graph*, J. Comput. System Sci., 46 (1993), pp. 91–128.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 721–737

THE GLAUBER DYNAMICS ON COLORINGS OF A GRAPH WITH HIGH GIRTH AND MAXIMUM DEGREE*

MICHAEL MOLLOY†

Abstract. We prove that the Glauber dynamics on the C-colorings of a graph G on n vertices with girth g and maximum degree Δ mixes rapidly if (i) $C = q\Delta$ and $q > q^*$, where $q^* = 1.4890\ldots$ is the root of $(1 - \mathrm{e}^{-1/q})^2 + q\mathrm{e}^{-1/q} = 1$; and (ii) $\Delta \geq D \log n$ and $g \geq D \log \Delta$ for some constant $D = D(q)$. This improves the bound of roughly 1.763Δ obtained by Dyer and Frieze [*Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, 2001] for the same class of graphs. Our bound on this class of graphs is lower than the bound of $11\Delta/6 \approx 1.833\Delta$ obtained by Vigoda [*J. Math. Phys.*, 41 (2000), pp. 1555–1569] for general graphs.

Key words. rapidly mixing Markov chains, Glauber dynamics, colorings

AMS subject classifications. 68W20, 05C15

DOI. 10.1137/S0097539702401786

Introduction. For a given graph G and integer C which is at least the chromatic number of G, we define the Glauber dynamics on the C-colorings of G to be the Markov chain described as follows. We start with an arbitrary C-coloring, and at each step we choose a uniformly random vertex v, and a uniformly random color c from $L(v)$, the list of colors which do not appear on any neighbors of v. Then we change the color of v to c. (If $L(v)$ is empty, then we do not choose a color; note that $L(v)$ is never empty if $C > \Delta$.)

Unless specified otherwise, we consider all colorings to be proper, i.e., no two adjacent vertices can have the same color. In section 3, it will be important to note that we can apply this step even to a nonproper coloring of G. But note that if a coloring is proper, then applying a step of the chain cannot produce an improper coloring.

The main question in this area is: For what values of C does this Markov chain mix in polytime? Usually this is studied in terms of Δ, the maximum degree of G. It is well known that for some graphs, the chain does not mix for $C \leq \Delta + 1$. (In fact, there are some graphs and $(\Delta + 1)$-colorings for which no color changes are possible, and so the chain is not even ergodic.) Jerrum [7] showed that for all graphs, the chain mixes in polytime for $C \geq 2\Delta$ and in optimal time, i.e., $O(n \log n)$ time, for $C \geq 2\Delta + 1$. Salas and Sokal [11] independently obtained the latter result. Vigoda [13] showed that for all graphs, a different chain mixes in optimal time for $C \geq \frac{11}{6}\Delta$ and this implies that for the same values of C, the Glauber dynamics mixes in polytime. Dyer, Greenhill, and Molloy [5] showed that the Glauber dynamics mixes in optimal time for $C \geq (2 - \epsilon)\Delta$, where ϵ is a small positive constant (see also [9]). Some work has been done on special classes of graphs. Dyer et al. [3] showed that the Glauber dynamics mixes in optimal time on triangle-free graphs when $C \geq (2 - x)\Delta$ for a different small positive constant x. More recently, Dyer and Frieze [2] showed that if the maximum degree, Δ, of G is at least $D \log n$ and the girth is at least $D \log \Delta$ for

*Received by the editors January 31, 2002; accepted for publication (in revised form) August 26, 2003; published electronically April 21, 2004. This work was supported by an NSERC research grant and a Sloan research fellowship.

http://www.siam.org/journals/sicomp/33-3/40178.html

†Department of Computer Science, University of Toronto, Toronto, ON, M4E 3G1 Canada (molloy@cs.toronto.edu).

some sufficiently large constant D, then the chain mixes in optimal time for $C \geq q\Delta$ for any constant $q > \beta$, where $\beta = 1.763\ldots < 11/6$ is the root of $\beta e^{-1/\beta} = 1$. Here we improve on this latter result. We define $q^* = 1.4890\ldots$ to be the root of $(1 - e^{-1/q})^2 + qe^{-1/q} = 1$.

THEOREM 1. *For any $q > q^*$ and integer ω, there exists D for which: Suppose G has n vertices, maximum degree $\Delta \geq D \log n$, and no vertex lies in more than ω cycles of length less than $D \log \Delta$. Then the Glauber dynamics mixes in time $O(n \log n)$ for $C = q\Delta$.*

Of course, this covers all graphs with girth at least $D \log \Delta$. It also covers natural models of random graphs such as $G_{n,p=c/n}$ for $c \approx D \log n$ and random Δ-regular graphs for $\Delta = D \log n$. Dyer and Frieze [2] showed that their theorem also extends to such graphs.

Throughout the paper, we assume that n is large enough for various asymptotic bounds to hold. We can also assume that q is sufficiently close to q^*. We use $N(v)$ to denote the neighborhood of v, i.e., the set of vertices which are adjacent to v. We define $d(v) = |N(v)|$ to be the degree of v. A *short cycle* is a cycle of length less than $D \log \Delta$.

1. Some intuition. The proofs of the results mentioned above, except for that of Vigoda [13], all come from the following idea.[1] Consider two colorings X, W which differ only at one vertex v. We will carry out one step of the process on each coloring, where we couple these two random steps maximally. Specifically, we first choose a uniform vertex u for both colorings. If $L_X(u)$ and $L_W(u)$ are the sets of allowable colors for u in X, W, respectively, then we take two mappings $f_X : [0,1] \to L_X(u), f_W : [0,1] \to L_W(u)$, such that

- for each $c \in L_X(u)$, $|f_X^{-1}(c)| = 1/|L_X(u)|$ and similarly for W, and
- $\{x : f_X(x) \neq f_W(x)\}$ is as small as possible.

Then we take a uniform random real $x \in [0,1]$ and choose $u, f_X(x)$ for X and $u, f_W(x)$ for W. Note that since $L_X(u), L_W(u)$ differ in at most one color per list, we will have $|\{x : f_X(x) \neq f_W(x)\}| \leq \min\{|L_X(u)|^{-1}, |L_W(u)|^{-1}\}$. Note further that $L_X(u) = L_W(u)$ unless u is a neighbor of v.

Using the path-coupling technique of Bubley and Dyer [1], it suffices to show that the probability of X, W converging after one step is greater than the probability of their differing in a second vertex after one step (we elaborate on this in section 3). They converge iff we choose $u = v$, which occurs with probability $1/n$. They differ on a second vertex iff we choose some $u \in N(v)$ and we choose $x \in \{x : f_X(x) \neq f_W(x)\}$. Since no list can ever be smaller than $C - \Delta$, this occurs with probability at most $\frac{\Delta}{n} \times \frac{1}{C-\Delta}$ which is less than $\frac{1}{n}$ so long as $C > 2\Delta$.

The bound was improved slightly in [3] by showing that for triangle-free graphs, after a relatively short period, most vertices will tend to have many repeated colors in their neighborhoods. Thus, their lists of available colors will tend to be somewhat greater than $C - \Delta$, and this leads to a gain in the above calculations. The same idea played a key role in [5].

In [2], Dyer and Frieze showed that for graphs with large girth and maximum degree, after $O(n \log n)$ steps, with high probability every vertex will have a list of size at least roughly $qe^{-1/q}\Delta$. If X, W are such that all vertices have lists of this size, then this yields that the probability of X, W differing on a second color is at most $\frac{\Delta}{n} \times \frac{1}{qe^{-1/q}\Delta}$. Since q is chosen so that $qe^{-1/q} > 1$, this probability is less than $\frac{1}{n}$.

[1] Jerrum's original proof in [7] predated this idea, but the idea yields a simpler proof.

The key new idea used in this paper is to show that, after $O(n \log n)$ steps, many neighbors u of v will satisfy $L_X(u) = L_W(u)$. If, for the sake of this intuitive introduction, we make the simplifying assumption that the graph is Δ-regular, then with high probability there will be roughly $(1 - \mathrm{e}^{-1/q})^2\Delta$ such neighbors. This improves our bound on the probability of X, W differing on a second color to $\frac{(1-(1-\mathrm{e}^{-1/q})^2)\Delta}{n} \times \frac{1}{q\mathrm{e}^{-1/q}\Delta}$, which is less than $\frac{1}{n}$ for $q > q^*$.

1.1. Uniform-like color sets. Suppose that G is Δ-regular and that every vertex in $N(v)$ is assigned an independent uniform color from $\{1, \ldots, C\}$. Then the probability that some color c does not appear on any neighbor of v is $(1 - 1/C)^\Delta = \mathrm{e}^{-1/q} + o(1)$. Thus we would expect that each list would have size roughly $C \times \mathrm{e}^{-1/q} = q\mathrm{e}^{-1/q}\Delta$. This explains, at least intuitively, the lower bound that Dyer and Frieze obtain.

Now suppose that every vertex of distance 2 from v is also assigned an independent uniform color from $\{1, \ldots, C\}$. Suppose further that we change the color of v in our coloring X to obtain another coloring W. For each $u \in N(v)$, $L_X(u) = L_W(u)$ iff the colors $X(v)$ and $W(v)$ both appear on $N(u) - v$. The probability that this occurs is $(1 - \mathrm{e}^{-1/q})^2 + o(1)$, and so this explains the result in this paper.

Of course, the colors appearing on the neighbors of v are far from independent. But intuitively, since there are few short cycles near v, after $O(n \log n)$ steps the colors on vertices close to v are "close enough" to being independent. Much of the work in this paper can be viewed as proving this statement.

1.2. A recursive analysis. Our situation is somewhat more complicated than that in [2]. To illustrate this, suppose that $N(v) = w_1, \ldots, w_\Delta$ and consider any assignment of colors to the vertices in $N(w_1), \ldots, N(w_\Delta)$. Now, assign to each w_i a uniform color from among those not appearing on $N(w_i)$, independently of the choice for any other w_j. It turns out, that for any assignment of colors to $N(w_1), \ldots, N(w_\Delta)$, the expected number of colors not appearing on $N(v)$ is at least $q\mathrm{e}^{-1/q}\Delta - o(\Delta)$. Thus, Dyer and Frieze did not have to ensure that the random colors appearing on $N(w_1), \ldots, N(w_\Delta)$ are close to uniform. Unfortunately, when analyzing our other parameter, we do not have this advantage and we need to prove rather tight results on the distributions of the colors appearing at distance 2 and 3 from v.

To do this, we require a complicated iterative analysis. We'll introduce that analysis now, in an oversimplified setting, before formalizing it in the next section. First, it will be much simpler to assume that G is regular and so every vertex has degree Δ. We will also pretend that the time steps are partitioned into a series of epochs, $\Psi_0, \Psi_1, \Psi_2, \ldots$ and that every vertex is recolored at least once during each epoch.

We will be interested in two parameters. The first is the number of colors available to each vertex. We will create a sequence $L_0 > L_1 > L_2 > \cdots$ such that during epoch Ψ_i, the set $L(v)$ of colors available to vertex v satisfies

$$|L(v)| \leq L_i$$

for every v. Furthermore, the same analysis as in [2] will yield that for $i \geq 1$, at any time during epoch Ψ_i we have

$$|L(v)| \geq (\mathrm{e}^{-1/q} - o(1))C = q\mathrm{e}^{-1/q}\Delta - o(\Delta).$$

The second parameter has to do with the probability that a particular color c is in L_v. For any color c and vertex v, we define

$$T(v,c) = \sum_{w \in N(v), c \in L(w)} \frac{1}{|L(w)|}.$$

Thus, if we were to choose a random color for each neighbor of v in turn, then $T(v,c)$ would be the expected number of neighbors which are assigned c. Furthermore, if we assume as in subsection 1.1 that these choices are independent, then the probability that no neighbors are assigned c, i.e., that at the end of these choices we have $c \in L(v)$, is roughly $\mathrm{e}^{-T(v,c)}$.

Of course, we need to be a bit careful here, because the sets $L(w)$ vary with time and we are actually interested in their values at different time steps. But we will overlook such details now as we are just providing an intuitive overview.

We will introduce sequences $A_0 < A_1 < A_2 < \cdots$ and $B_0 > B_1 > B_2 > \cdots$ such that during epoch Ψ_i, we have for every pair v, c

$$A_i \le T(v,c) \le B_i.$$

We start with our recursive equation for A_i. Suppose that we are in epoch Ψ_{i+1} and for each neighbor w of v, let's pretend that the neighbors of w were most recently colored during epoch Ψ_i. At that time we had $T(w,c) \le B_i$ and so the probability that c is now in $L(w)$ is at least roughly e^{-B_i}. We also have $|L(w)| \le L_i$. This inspires us to define

$$A_{i+1} = \Delta \times \mathrm{e}^{-B_i}/L_i.$$

Similarly, we obtain

$$B_{i+1} = \Delta \times \mathrm{e}^{-A_i}/(\mathrm{e}^{-1/q}C).$$

Finally, we consider L_{i+1}. $\mathbf{Exp}(|L(v)|)$ is roughly $\sum_{c=1}^{C} \mathrm{e}^{-T(v,c)}$. Also, note that

$$\sum_{c=1}^{C} T(v,c) = \sum_{w \in N(v)} \sum_{c \in L(w)} \frac{1}{|L(w)|} = \Delta.$$

Thus, our expression for $\mathbf{Exp}(|L(v)|)$ is minimized when each $T(v,c)$ is equal to Δ/C, which yields a nonrecursive lower bound of $C \times \mathrm{e}^{-\Delta/C} = C \times \mathrm{e}^{-1/q}$, as obtained in [2]. Furthermore, the expression is maximized when the values of $T(v,c)$ are as disparate as possible. Since each $T(v,c)$ is between A_i and B_i, we get a recursive upper bound by assuming that every $T(v,c)$ is either A_i or B_i. Since they sum to Δ, we must have $(\Delta - CB_i)/(A_i - B_i)$ of them equal to A_i and $(\Delta - CA_i)/(B_i - A_i)$ of them equal to B_i. That gives an upper bound of

$$L_{i+1} = \frac{\Delta - CB_i}{A_i - B_i} \times \mathrm{e}^{-A_i} + \frac{\Delta - CA_i}{B_i - A_i} \times \mathrm{e}^{-B_i}.$$

After choosing appropriate initial values, it is straightforward to show that these recursive equations have a limit of $A_i = B_i = 1/q$ and $L_i = C \times \mathrm{e}^{-1/q}$. Therefore, by running our Markov chain for enough epochs, we can guarantee that for every v, c we have $L(v)$ arbitrarily close to $\mathrm{e}^{-1/q}$ and $T(v,c)$ arbitrarily close to Δ/C. This

goes a long way towards allowing us to show that the intuitive analysis outlined in subsection 1.1 holds.

To transform this intuition into a proof, we need to be much more precise. For one thing, we have to be careful about specifying the time steps at which we are measuring $L(v)$ in some of these quantities. We also can't assume that every vertex is selected exactly once per epoch; it turns out that at least once per epoch will do, and we can achieve that by taking each epoch to be of length $O(n \log n)$. Also, these recursive equations were obtained by (implicitly) assuming that in each epoch, every quantity will be equal to its expected value. In order to allow for the possibility that some quantities differ slightly from their expected values, we decrease each A_i and increase each B_i, L_i by a small amount; we also apply a concentration inequality to show that they don't differ more than slightly from their expected values. Finally, we can't assume that every vertex has degree exactly Δ, and so we have to allow our bounds on $|L(v)|$ to be functions of the degree of v.

All of the adjustments outlined in the preceding paragraph are straightforward, but tedious if we assume, as in subsection 1.1, that the random colors assigned to the neighbors of a vertex are independent. Of course, this assumption is not valid and so we need to prove that they are, in some sense, close to being independent. We do this by focusing on "long paths of disagreement," which will be described further in later sections.

2. The main lemmas. As described in the opening part of section 1, our goal is essentially to show that after $O(n \log n)$ steps, many neighbors u of v will satisfy $L_X(u) = L_W(u)$. We must complicate this condition somewhat. First, for technical reasons, we wish to make this hold independently of our choice of the colors v has in X, W and so we make a statement that holds for every pair of colors c_1, c_2. Also, we have to adjust our goal somewhat to deal with the case where v has neighbors of degree less than Δ. In fact, neighbors of very small degree, less than $\rho\Delta$ for some small positive constant ρ, are particularly problematic and so have to be dealt with separately. Our main lemma is as follows.

LEMMA 2. *For every $\epsilon, \rho > 0$, there exist constants D, τ such that with probability at least $1 - O(n^{-6})$, for every vertex v, colors c_1, c_2, and time $\tau n \log n \le t \le n^2$, we have the following: Define $\theta = \theta_{c_1,c_2}(v)$ to be the set of neighbors w of v with $d(w) \ge \rho\Delta$ and with at least one of c_1, c_2 not appearing in $N(w) - v$, and define*

$$R_{c_1,c_2}(v) = \sum_{w \in \theta} \frac{1}{|L(w)|}.$$

Then

$$R_{c_1,c_2}(v) \le \frac{1 - (1 - \mathrm{e}^{-1/q})^2}{q\mathrm{e}^{-1/q}} \times \frac{d(v)}{\Delta} + \epsilon.$$

Note that, under the notation of section 1, if v has c_1 in X and c_2 in W, and if $d(w) > \rho\Delta$ and $L_X(w) \ne L_W(w)$, then $w \in \theta_{c_1,c_2}(v)$. Thus if the graph is Δ-regular, then this lemma implies what we said our goal was in the opening part of section 1.

In section 3, we will strengthen this lemma and then show how it implies Theorem 1. For ease of presentation, we first prove Lemma 2 and then show how to adapt the proof to yield the stronger lemma.

We begin with a recursive definition. This is along the same lines as that described in subsection 1.2, but modified somewhat to facilitate a formal proof. For

each $0 \le d \le \Delta$,

- $\alpha_1 = 0, \beta_1 = 1/(q-1), \lambda_1^{(d)} = q$;
- $\alpha_{k+1} = \mathrm{e}^{-\beta_k}/\lambda_k^{(\Delta)}$;
- $\beta_{k+1} = \mathrm{e}^{-\alpha_k}/(q\mathrm{e}^{-1/q})$;
- $\lambda_{k+1}^{(d)} = \frac{q\beta_k - 1}{\beta_k - \alpha_k}\mathrm{e}^{-\alpha_k d/\Delta} + \frac{1-q\alpha_k}{\beta_k-\alpha_k}\mathrm{e}^{-\beta_k d/\Delta}$.

LEMMA 3. $\lim_{k\to\infty}\alpha_k = \lim_{k\to\infty}\beta_k = 1/q, \lim_{k\to\infty}\lambda_k^{(d)} = q\mathrm{e}^{-d/q\Delta}$.

We postpone the proof of Lemma 3 until later. But for now, we require the following simple observation.

LEMMA 4. *For all $k \ge 1$ and $0 \le d \le \Delta$,*

(a) $\alpha_k < 1/q < \beta_k$, *and*

(b) $q\mathrm{e}^{-\beta_k d/\Delta} \le \lambda_{k+1}^{(d)} \le q\mathrm{e}^{-\alpha_k d/\Delta}$.

Proof. (b) follows since $\lambda_{k+1}^{(d)}$ is a linear combination of $q\mathrm{e}^{-\beta_k d/\Delta}$ and $q\mathrm{e}^{-\alpha_k d/\Delta}$. (a) follows from a simple induction. □

In subsection 1.2 we said that we have to adjust our sequences by a small amount, in order to allow for the possibility that some terms differ slightly from their expected values. We do so now.

Fix some small δ to be named later, and choose k^* such that $\alpha_{k^*} > \frac{1}{q} - \delta/2$, $\beta_{k^*} < \frac{1}{q} + \delta/2$, and $\lambda_{k^*} < q\mathrm{e}^{-1/q} + \delta/2$ (such a k^* exists by Lemma 3). Choose a sufficiently small constant ζ and constants $a_1, \dots, a_{k^*}, b_1, \dots, b_{k^*}, \ell_1^{(0)}, \dots, \ell_{k^*}^{(\Delta)}$ such that

(i) $a_1 = \alpha_1, b_1 = \beta_1, \ell_1^{(d)} = \lambda_1^{(d)}$;

(ii)
- $a_{k+1} < \mathrm{e}^{-b_k}/\ell_k^{(\Delta)} - \zeta$;
- $b_{k+1} > \mathrm{e}^{-a_k}/(q\mathrm{e}^{-(1+\zeta)/q}) + \zeta$;
- $\ell_{k+1}^{(d)} = \frac{qb_k - 1}{b_k - a_k}\mathrm{e}^{-(1-\zeta)a_k d/\Delta} + \frac{1-qa_k}{b_k-a_k}\mathrm{e}^{-(1-\zeta)b_k d/\Delta}$

(iii) $a_{k^*} > \frac{1}{q} - \delta, b_{k^*} < \frac{1}{q} + \delta$.

The existence of such constants follows easily from the continuity of the relevant functions over the region $a \ne b$.

Note that we can assume that ρ is as small as we wish, as if Lemma 2 holds with $\rho = \rho_1$ then it clearly holds with $\rho = \rho_2$ for any $\rho_2 > \rho_1$. In particular, we will assume that

- $1/(q-\rho) < \mathrm{e}^{(1+\zeta-a_{k^*})/q}/q$; and
- $\mathrm{e}^{-\rho/(q-1)} > \mathrm{e}^{1/q - b_{k^*}}$.

We define $L_t(v)$ to be the set of colors available for v at time t; we sometimes omit the subscript t when it is not necessary. For any vertex u and time t, we define $\bar{t}(u,t)$ to be the last time before t that u is selected. To be precise, we define $\bar{t}(u,t)$ to be zero in the event that u is not selected before time t. This notation allows us to define a more careful refinement of the vague quantity $T(v,c)$ from subsection 1.2:

$$T_t(v,c) = \sum_{w\in N(v), c\in L_{\bar{t}(w,t)}(w)} \frac{1}{|L_{\bar{t}(w,t)}(w)|}.$$

Note that

$$\sum_{c=1}^{C} T_t(v,c) = \sum_{w\in N(v)} \sum_{c\in L_{\bar{t}(w,t)}(w)} \frac{1}{|L_{\bar{t}(w,t)}(w)|} = d(v).$$

We will prove the following lemma inductively.

LEMMA 5. *For each $1 \le k \le k^*$, with probability at least $1 - n^{-6}$, for every v with $d(v) > \rho\Delta$, color c and $30kn \log n \le t \le n^2$, we have*

(a) $q e^{-(1+\zeta)d(v)/C}\Delta \le |L_t(v)| \le \ell_k^{(d(v))}\Delta$;

(b) $a_k d(v)/\Delta \le T_t(v,c) \le b_k d(v)/\Delta$.

The lower bound in part (a) is essentially Lemma 4.1 and (7) of Dyer and Frieze [2], and our proof is similar to theirs.

Proof. The proof is by induction on k. The base case $k = 1$ holds trivially. So assume that it holds for some k and consider $k+1$.

Let the neighbors of v be $w_1, \dots, w_{d(v)}$ and for each i let $t_i = \bar{t}(w_i, t)$ be the last time before t that w_i is selected. For each i, let $N_{w_i} - v = \{w_{i,1}, \dots, w_{i,d(w_i)-1}\}$ and let $t_{i,j} = \bar{t}(w_{i,j}, t_i)$ be the last time before t_i that $w_{i,j}$ is selected. Similarly, for each i, j let $N_{w_{i,j}} - w_i = \{w_{i,j,1}, \dots, w_{i,j,d(w_{i,j})-1}\}$ and let $t_{i,j,r} = \bar{t}(w_{i,j,r}, t_{i,j})$ be the last time before $t_{i,j}$ that $w_{i,j,r}$ is selected. (If v lies in some cycles of length at most 6, then some vertices will receive more than one label, but this does not create a problem.)

The idea of focusing on the colors assigned at these times was used by Dyer and Frieze [2].

With high probability, each $t_i \ge t - 10n\log n$. Indeed, the probability that this is not true is at most $\Delta(1 - \frac{1}{n})^{10n\log n} < n^{-9}$. The same calculations give the same bound on the probability that some $t_{i,j}$ is less than $t - 20n\log n$ or that some $t_{i,j,r}$ is less than $t - 30n\log n$.

We start by proving part (a). Expose the values of $t_1, \dots, t_{d(v)}, t_{1,1}, \dots, t_{d(v),d(w_{d(v)})-1}$.

It would be nice if we could say that the colors assigned to $w_1, \dots, w_{d(v)}$ were independent, as that would greatly simplify our calculations. However, this is clearly not the case, since the color assigned to w_i has an effect on the next color assigned to a neighbor of w_i, and this effect can propagate along a path which eventually leads to some w_j. This can be a very short path if it goes through v; we will deal with such paths later. Otherwise, unless w_i is one of the at most $O(1)$ vertices lying on a short cycle through v, the path must have length at least $D\log\Delta - 2$. Our first concern will be such long paths.

Consider the following procedure, which we denote GLAUB(i). It follows the usual Glauber dynamics, but after step t_i all neighbors of w_i ignore w_i. More specifically, for each $u \in N(w_i)$, $L(u)$ is the set of colors which do not appear on $N(u) - w_i$, and whenever u is selected, u is assigned a uniformly random color from $L(u)$. Thus, for example, w_i might have the same color as some of its neighbors.

We use GLAUB to denote the usual Glauber dynamics, and we consider the two procedures to be coupled in that each has the same initial state and chooses the same vertex at each step. The color choice at each step is coupled maximally.

A *long path of disagreement* from w_i is a path P of length at least $\ell = (D/3)\log\Delta$ beginning at w_i and not going through v, such that the color of each vertex in the path differs between GLAUB and GLAUB(i). (Such paths were used in a slightly different way by Dyer and Frieze [2] and earlier, in a different setting, by van den Berg and Steif [12].) We say that w_i is *influential* if there is a long path of disagreement from w_i or if by changing the color assigned to w_i at time t_i, and not changing any future color/vertex choices in GLAUB(i), it is possible to create a long path of disagreement from w_i.

We define B to be the set of neighbors w_i such that either w_i lies in a cycle through v of length less than $D\log\Delta$ or w_i is influential.

LEMMA 6. *With probability at least* $1 - n^{-10}$, $|B| < \Delta/\log\Delta$.

Proof. The set of neighbors which lie on a short cycle through v is at most 2ω. Therefore, we only have to count the influential vertices which do not lie on any short cycle through v. Consider such a vertex w_i; we will bound the probability that it is influential.

We will bound this probability by the probability that either w_i is influential, or $t_i < t - 20n\log\Delta$. The probability of the latter is $(1-1/n)^{20n\log\Delta} < \frac{1}{2}\Delta^{-11}$.

If w_i is influential and $t_i \geq t - 20n\log\Delta$, then there is a path $P : w_j = p_0, p_1, \ldots, p_\ell$, a color c to assign to w_i at time t_i, and time steps $t - 20n\log\Delta < s_1 < \cdots < s_\ell \leq t$ such that at step s_r, p_r is selected and assigned different colors in the two procedures.

Suppose that P is the lexicographically first such path formed. Thus, at each step s_r, if the colors of some neighbor u of p_r disagree in the two procedures, then there is a path of disagreement from u to w_j which does not go through p_r. Since that path has length less than ℓ, then p_r and u lie in a cycle of length less than $D\log\Delta$. Since p_r lies in at most ω such cycles, there can be at most 2ω such neighbors. Thus, since $|L(p_r)|$ is always at least $C - \Delta = (q-1)\Delta$, the probability that p_r chooses a different color for each procedure, under the maximal coupling, is less than $2\omega/((q-1)\Delta)$. Therefore, the probability that such a path is formed is at most

$$\Delta \times (\Delta-1)^\ell \times \binom{20n\log\Delta}{\ell} \times \left(\frac{2\omega}{(q-1)\Delta}\right)^\ell < \Delta \times \left(\frac{100\mathrm{e}\omega}{D}\right)^\ell < \frac{1}{2}\Delta^{-11}$$

for D sufficiently large and q near q^*. Thus, after adding the probability that $t_i < t - 20n\log\Delta$, the probability that w_i is influential is at most Δ^{-11}.

Now consider any collection $w_{i_1}, \ldots, w_{i_m}$ of $m = \Delta/\log\Delta - 2\omega$ neighbors which don't lie in short cycles through v. Using the fact that long paths of disagreement from any two such neighbors must be disjoint, similar calculations yield that the probability of all m neighbors being influential is at most $(\Delta^{-11})^m$. Therefore, the probability that at least m such neighbors of v are influential is at most

$$\binom{\Delta}{m} \times \Delta^{-11m} < \Delta^{-10m} < n^{-10}$$

for $D > 1$. ∎

Now, for each i, j, we expose the color assigned to $u_{i,j}$ at time $t_{i,j}$. We also expose the set B. We denote this set of information, along with the values of $t_1, \ldots t_{d(v)}, t_{1,1}, \ldots t_{d(v),d(w_{d(v)})-1}$, by $\mathcal{H}$. We say that $\mathcal{H}$ is *good* if every $t_i, t_{i,j} \geq t - 30n\log n$, if the sets of color assignments satisfy the conditions of Lemma 5 for k, and if $|B| \leq \Delta/\log\Delta$. We will show by induction that the probability of $\mathcal{H}$ not being good is at most $O(n^{-9})$.

LEMMA 7. *For any good* $\mathcal{H}$*, the conditional probability that, at a particular time* $t \geq 30(k+1)n\log n$ *during MOD-GLAUB,* $|L(v)| \leq q\mathrm{e}^{-(1+\zeta/2)d(v)/C}\Delta$ *or* $|L(v)| \geq \ell_{k+1}^{(d(v))}\Delta - \zeta/2$ *is at most* n^{-9}.

Proof. Rather than analyzing $|L(v)|$ directly, we will focus on $|L^*(v)|$, the set of colors which do not appear on any $w_i \in N(v) - B$. Note that $|L^*(v)| \geq |L(v)| \geq |L^*(v)| - |B| = |L^*(v)| - o(\Delta)$.

Rather than dealing with GLAUB directly, we consider the procedure MOD-GLAUB, whereby after time $t - 30n\log n$, neighbors of v ignore the color on v.

The advantage of focusing on $|L^*(v)|$ and MOD-GLAUB is that the color assignments to the vertices in $N(v) - B$ are independent. This is because no combination of assignments to these vertices can produce a long path of disagreement from one such vertex to another, and, since the vertices ignore the color on v and no two lie on a short cycle through v, there are no short paths of disagreement between these vertices.

Furthermore, exposing the fact that $w_i \notin B$ only exposes that w_i is not influential. Since the definition of "influential" does not depend on the particular color assigned to w_i, this does not expose anything about that choice of color.

So each w_i receives a uniformly random color from among those colors which $\mathcal{H}$ dictates to not appear on $N(w_i) - v$ at time t_i, and the choices for $w_1, \dots, w_\Delta$ are independent. Thus, for any color c the conditional probability that c belongs to $L^*(v)$ at time t is

$$\prod_{w \in N(v)-B, c \in L(w)} 1 - \frac{1}{L(w)} = \exp(-T(v,c)) + o(1),$$

since every $L(w)$ has size at least $C - \Delta = (q-1)\Delta$, and since $|B| = o(\Delta)$. Therefore, the expected size of $L^*(v)$ at time t is equal to $\sum_c \exp(-T(v,c)) + o(\Delta)$.

Recall that $\sum_c T(v,c) = d(v)$. Furthermore, since $\mathcal{H}$ is good, each $T(v,c)$ lies between $a_k d(v)/\Delta$ and $b_k d(v)/\Delta$. Subject to these constraints, $\sum_c \exp(-T(v,c))$ is easily seen to be minimized when every $T(v,c) = d(v)/C$, and maximized when every $T(v,c)$ is either $a_k d(v)/\Delta$ or $b_k d(v)/\Delta$. In the latter case, the fact that there are $C = q\Delta$ different $T(v,c)$ terms and they sum to $d(v)$ implies that $\Delta(qb_k - 1)/(b_k - a_k)$ of them are equal to $a_k d(v)/\Delta$ and the remaining $\Delta(1 - qa_k)/(b_k - a_k)$ of them are equal to $b_k d(v)/\Delta$. This yields

$$q\mathrm{e}^{-d(v)/C}\Delta \le \mathbf{Exp}(|L^*(v)|) \le \Delta \times \left(\frac{qb_k - 1}{b_k - a_k} \mathrm{e}^{-a_k d(v)/\Delta} + \frac{1 - qa_k}{b_k - a_k} \mathrm{e}^{-b_k d(v)/\Delta} \right).$$

Note that these calculations can be used to show that for all d, k,

$$\ell_k^{(d)} > q\mathrm{e}^{-d/q}. \tag{1}$$

Since the color choice for w_i can affect $|L^*(v)|$ by at most 1, Azuma's inequality implies that $|L^*(v)|$ is highly concentrated and, in particular, that the probability of it differing from its expected value by $\Theta(\Delta)$ is $\mathrm{e}^{-\Theta(\Delta)}$. Since $d(v) > \rho\Delta$, this proves our bound on $|L^*(v)|$ in MOD-GLAUB.

Now we extend this bound to GLAUB. If the color of w_i differs in the two procedures, then at some step after $t - 30n \log n$, w_i is assigned a color that appears on v and on no other neighbor of w_i. Since $|L(w_i)|$ is always greater than $C - \Delta = (q-1)\Delta$, this occurs with probability at most $((q-1)n\Delta)^{-1}$ at any one time step, so the probability that it occurs at least once is at most $30n \log n/((q-1)n\Delta) < 100/D$.

So the expected number of vertices in $N(v)$ which are affected in this way is at most $100\Delta/D = O(\log n)$. A simple application of the Chernoff bounds shows that this number is highly concentrated, and so the probability that it is higher than $\Theta(\Delta)$ is at most $\mathrm{e}^{-\Theta(\Delta)} < n^{-10}$ for $\Delta \ge D \log n$, where D is sufficiently large in terms of ν. This proves the lemma. □

Adding the probability of $n^{-10} + O(n^{-9})$ that $\mathcal{H}$ is not good and multiplying by the $n \times n^2$ choices for v, t establish part (a) of Lemma 5.

Part (b) follows in the same manner. We define B^* in a similar way to B, with the exception that a neighbor w_i is in B^* if (i) $w_i \in B$, (ii) it is possible, by changing the color assigned to a neighbor u of w_i, to form a long path of disagreement from u not passing through w_i, or (iii) some neighbor u of w_i lies in a short cycle through w_i. The same analysis shows that $\mathbf{Pr}(|B^*| > \Delta/\log\Delta) < n^{-11}$. (An extra factor of Δ appears in the expected number calculation, and this is not enough to raise that expected number significantly.) We then restrict our attention to

$$T_t^*(v,c) = \sum_{w \in N(v) - B*, c \in L_{\bar{t}(w,t)}(w)} \frac{1}{|L_{\bar{t}(w,t)}(w)|}.$$

$\mathcal{H}$ exposes B^*, all times $t_i, t_{i,j}, t_{i,j,r}$, and the colors assigned to each $u_{i,j,r}$ at time $t_{i,j,r}$. Then we modify our procedure as follows: All vertices ignore the colors on $v, w_1, \ldots, w_\Delta$, and after time $t_{i,j}$, the neighbors of $w_{i,j}$ ignore the color of $w_{i,j}$. If some $w_{i,j}$ is adjacent to $w_{i',j'}$, then they ignore each other's colors. We refer to this modified procedure as MOD-GLAUB2.

Consider any $w_i \notin B^*$ and let $I(w_i, c)$ be the indicator variable that $c \in L(w_i)$ at time t_i. Suppose that $d(w_i) > \rho\Delta$. As in the proof of part (a), $\mathbf{Pr}(I(w_i,c) = 1) = \exp(-T_{t_i}(w_i,c)) + o(1)$. Therefore, using the bounds on $T_{t_i}(w_i,c)$ and the list sizes from the fact that $\mathcal{H}$ is good, the expected value of $I(w_i,c)/|L_{t_i}(w_i)|$ is at most

$$(\mathrm{e}^{-a_k d(w_i)/\Delta} + o(1)) \times \frac{1}{q\mathrm{e}^{-(1+\zeta)d(w_i)/C}\Delta} \le \frac{1}{\Delta} \times \mathrm{e}^{-a_k/q}/(q\mathrm{e}^{-(1+\zeta)/q}) + o(1),$$

since the fact that $a_k < 1/q$ implies that the left-hand side is maximized at $d(w_i) = \Delta$.

We must take more care in proving our lower bound. We use L to denote $|L(w_i)|$ at time t_i, we let R be the event that $L \le \ell_{k+1}^{(d(v))}\Delta$, and we let I_R be the indicator variable for R. In part (a), we proved that $\mathbf{Pr}(I_R = 0) = O(n^{-9})$. Therefore,

$$\begin{aligned}
\mathbf{Exp}\left(\frac{I(w_i,c)}{L}\right) &\ge \mathbf{Exp}\left(\frac{I(w_i,c)}{L} \times I_R\right) \\
&\ge \frac{\mathbf{Pr}(I(w_i,c)=1) - \mathbf{Pr}(I_r = 0)}{\ell_{k+1}^{(d(v))}\Delta} \\
&= \frac{\mathbf{Pr}(I(w_i,c))}{\ell_{k+1}^{(d(v))}\Delta} + o(1/\Delta).
\end{aligned}$$

Furthermore, we have

$$\begin{aligned}
\ell_{k+1}^d &= \frac{1}{b_k - a_k}\left((qb_k - 1)\mathrm{e}^{-(1-\zeta)a_k d/\Delta} + (1 - qa_k)\mathrm{e}^{-(1-\zeta)b_k d/\Delta}\right) \\
&< \frac{1}{b_k - a_k}\left((qb_k - 1)\mathrm{e}^{-(1-\zeta)a_k} + (1 - qa_k)\mathrm{e}^{-(1-\zeta)b_k}\right) \times \mathrm{e}^{(1-\zeta)b_k(1-d/D)} \\
&= \ell_{k+1}^{(\Delta)} \times \mathrm{e}^{(1-\zeta)b_k(1-d/D)}.
\end{aligned}$$

Thus, the expected value of $I(w_i, c)/|L(w_i)|$ at time t is at least

$$\begin{aligned}(\mathrm{e}^{-b_k d(w_i)/\Delta} + o(1)) \times \frac{1}{\ell_{k+1}^{(d(w))}\Delta} &\geq \frac{1}{\Delta} \times \frac{\mathrm{e}^{-b_k}}{\ell_{k+1}^{(\Delta)}} \times \mathrm{e}^{(b_k-(1-\zeta)b_k)(1-d(w_i)/\Delta)} + o(1)\\ &\geq \frac{1}{\Delta} \times \frac{\mathrm{e}^{-b_k}}{\ell_{k+1}^{(\Delta)}} + o(1)\\ &\geq \frac{1}{\Delta} \times \frac{\mathrm{e}^{-b_k}}{\ell_{k}^{(\Delta)}} + o(1).\end{aligned}$$

If $d(w_i) \leq \rho\Delta$, then $|L(w_i)|$ is never less than $C - \rho\Delta = (q-\rho)\Delta$. Therefore, by our assumptions on how small ρ is, the expected value of $I(w_i, c)/|L(w_i)|$ is at most

$$\frac{1}{(q-\rho)\Delta} < \frac{1}{\Delta} \times \mathrm{e}^{-a_k/q}/(q\mathrm{e}^{-(1+\zeta)/q}).$$

Furthermore, since no list is ever smaller than $(q-1)\Delta$, $\mathbf{Pr}(I(w_i, c)) \geq (1 - \frac{1}{(q-1)\Delta})^{\rho\Delta}$ which, by (1) and our assumptions on the size of ρ, is greater than $\frac{C}{\Delta} \times \mathrm{e}^{-b_k}/\ell_k^{(\Delta)}$. Therefore, the expected value of $I(w_i, c)/|L(w_i)|$ is again at least $\frac{1}{\Delta} \times \mathrm{e}^{-b_k}/\ell_k^{(\Delta)}$.

This implies that the expected value of $T_t^*(v, c)$ is at most $(a_{k+1} - \zeta)d(v)/\Delta$ and at least $(b_{k+1} - \zeta)d(v)/\Delta$. By viewing the color assignments to $w_{i,1}, \ldots, w_{i,\Delta-1}$ as one single random choice, we have Δ choices, each of which can affect $T_t^*(v, c)$ by at most $1/((q-1)\Delta)$, since no $L(u)$ can have size less than $(q-1)\Delta$. Therefore by Azuma's inequality, the probability that, under the procedure MOD-GLAUB2, $T_{v,c}$ differs from its expected value by more than $(\zeta/2)d(v)/\Delta$ is at most $\mathrm{e}^{-\Theta(\Delta)} < n^{-10}$ for D sufficiently large in terms of ζ. (Again, we use the fact that $d(v) \geq \rho\Delta$.)

Virtually the same argument as that used for part (a) proves that with sufficiently high probability, $T_t^*(v, c)$ in GLAUB is within $(\zeta/4)d(v)/C$ of its value in MOD-GLAUB2. Furthermore, if $\mathcal{H}$ is good then $|T_t^*(v, c) - T_t(v, c)| < (\Delta/\log\Delta) \times (1/(q-1)\Delta) = o(1)$. This proves part (b).

Finally, we need to note that, since with probability at least $1 - n^{-9}$ we have each $t_i, t_{i,j} \geq t - 30n\log n \geq 30kn\log n$, then we have by induction that the probability $\mathcal{H}$ is not good is at most $2n^{-9}$. □

We now show that Lemma 2 follows from Lemma 5 if we take δ sufficiently small when specifying k^* and thus obtain $a_k, b_k, \ell_{k^*}^d$ sufficiently close to $1/q, 1/q, q\mathrm{e}^{-d/C}$.

Proof of Lemma 2. We prove Lemma 2 in the same way as the inductive step for Lemma 5(b). We consider the procedure MOD-GLAUB2. We expose $\mathcal{H}$ and, by taking $\tau \geq 30k^*$, we can assume it is (with probability at least $1 - O(n^{-6})$) such that for each w_i with $d(w_i) > \rho\Delta$,

$$\left(\frac{1}{q} - \delta\right) d(w_i)/\Delta < T(w_i, c_1), T(w_i, c_2) < \left(\frac{1}{q} + \delta\right) d(w_i)/\Delta,$$

and

$$|L(w_i)| \geq q\mathrm{e}^{-(1+\zeta)d(v)/C}\Delta$$

and that $|B^*| < \Delta/\log\Delta$. This last assumption, along with the fact that $|L(u)| \geq C - q$ for every vertex u, implies that $\sum_{w_i \in B^*} |L(w_i)|^{-1} = o(1)$. Thus we can restrict our attention to $N(v) - B^*$.

Now consider any $w_i \in N(v) - B^*$ with $d(w_i) > \rho\Delta$. Let E_1 (resp., E_2) be the event that c_1 (resp., c_2) appears on $N(w_i) - v$. Thus $E_1 \cap E_2$ is the event that $w_i \notin \theta$. We will estimate $\mathbf{Pr}(E_1 \cap E_2) = 1 - \mathbf{Pr}(\overline{E_1}) - \mathbf{Pr}(\overline{E_2}) + \mathbf{Pr}(\overline{E_1} \cap \overline{E_2})$. By our assumption on $\mathcal{H}$, each $\mathbf{Pr}(\overline{E_r})$ is at most

$$\exp(-T(w_i, c_r)) \leq \exp\left(-\left(\frac{1}{q} - \delta\right) d(w_i)/\Delta\right).$$

Also, $\mathbf{Pr}(\overline{E_1} \cap \overline{E_2})$ is at least

$$\begin{aligned}
&\prod_{u \in N(w_i); c_1 \in L(u); c_2 \notin L(u)} \left(1 - \frac{1}{|L(u)|}\right) \times \prod_{u \in N(w_i); c_2 \in L(u); c_2 \notin L(u)} \left(1 - \frac{1}{|L(u)|}\right) \\
&\quad \times \prod_{u \in N(w_i); c_1, c_2 \in L(u)} \left(1 - \frac{2}{|L(u)|}\right) \\
&= \exp(-T(w_i, c_1) - T(w_i, c_2)) + o(1) \\
&\geq \exp\left(-2\left(\frac{1}{q} + \delta\right) d(w_i)/\Delta\right) + o(1).
\end{aligned}$$

For δ sufficiently small in terms of ϵ, this yields $\mathbf{Pr}(E_1 \cap E_2) > (1 - \mathrm{e}^{-d(w_i)/q\Delta})^2 - \epsilon/2$. Setting $y = \mathrm{e}^{-d(w_i)/q\Delta}$, note that $(1 - (1-y)^2)/(yq\Delta) = (2-y)/(q\Delta)$ increases as y decreases and so is maximized at $d(w_i) = \Delta$. Therefore, for ζ sufficiently small in terms of ϵ, we have

$$\begin{aligned}
\mathbf{Exp}(R_{c_1,c_2}(v)) &\leq o(1) + \sum_{w \in N(v)} \frac{(1 - (1 - \mathrm{e}^{-d(w_i)/q\Delta})^2) + \epsilon/2}{q\mathrm{e}^{-(1+\zeta)d(w_i)/C}\Delta} \\
&\leq \epsilon/2 + \sum_{w \in N(v)} \frac{(1 - (1 - \mathrm{e}^{-d(w_i)/q\Delta})^2)}{q\mathrm{e}^{-d(w_i)/C}\Delta} \\
&\leq \epsilon/2 + \frac{(1 - (1 - \mathrm{e}^{-1/q})^2)}{q\mathrm{e}^{-1/q}} \times \frac{d(v)}{\Delta}.
\end{aligned}$$

It follows as in the proof of Lemma 5(b) that this sum is highly concentrated and so the probability that it differs from its expected value by more than $\epsilon/4$ is at most n^{-10} for D sufficiently large.

It follows again as in the proof of Lemma 5(b) that the probability of it differing by more than $\epsilon/4$ from GLAUB to MOD-GLAUB is at most n^{-10} for D sufficiently large. This proves Lemma 2.

We close this section with the proof of Lemma 3, thus completing the proof of Lemma 2.

Proof of Lemma 3. Recall that we can assume that $q > q^*$ is sufficiently small, and so we will take $q < 1.49$.

It is straightforward to show that α_k is strictly increasing and β_k is strictly decreasing, and so by Lemma 2, $\alpha = \lim_{k\to\infty} \alpha_k, \beta = \lim_{k\to\infty} \beta_k, \lambda = \lim_{k\to\infty} \lambda_k^{(\Delta)}$ exist. They must satisfy

$$\begin{aligned}
\alpha &= \mathrm{e}^{-\beta}/\lambda, \\
\beta &= \mathrm{e}^{-\alpha}/(q\mathrm{e}^{-1/q}), \\
\lambda &= \frac{q\beta - 1}{\beta - \alpha}\mathrm{e}^{-\alpha} + \frac{1 - q\alpha}{\beta - \alpha}\mathrm{e}^{-b}.
\end{aligned}$$

We will prove that this system has no roots for $0 \le \alpha < 1/q$. This, along with Lemma 4, implies Lemma 3.

Rearranging the first equation of our system, we get $f(\alpha) = \alpha\lambda - \mathrm{e}^{-\beta} = 0$. We will bound the derivative of f with respect to α. We start by bounding

$$g(\alpha) = 1 - \beta q + \beta - \alpha\beta q.$$

Clearly $g(\alpha) \to 0$ as $\alpha \to 1/q$. Also, noting that the derivative of β with respect to α is $-\beta$, we have $g'(\alpha) = q\beta - \beta - q\beta + q\alpha\beta = q\alpha\beta - \beta < 0$ for $\alpha < 1/q$. Therefore, $g(\alpha) > 0$ for $\alpha < 1/q$.

Now, using the fact that by (1) $q\mathrm{e}^{-1/q} \le \lambda \le q\mathrm{e}^{-\alpha}$, we have

$$\begin{aligned}
f'(\alpha) &= \lambda - \beta\mathrm{e}^{-\beta} + \frac{\alpha}{\beta-\alpha}((1-2\beta q)\mathrm{e}^{-\alpha} + (\beta - q - \alpha\beta q)\mathrm{e}^{-\beta} + (\beta+1)\lambda) \\
&\ge \lambda - \beta\mathrm{e}^{-\beta} + \frac{\alpha\mathrm{e}^{-\beta}}{\beta-\alpha}(1 - 2\beta q + \beta - q - \alpha\beta q + q\beta + q) \\
&\qquad + \frac{\mathrm{e}^{-\alpha} - \mathrm{e}^{-\beta}}{\beta-\alpha}(\alpha - 2\alpha\beta q) \\
&\ge \lambda - \beta\mathrm{e}^{-\beta} + \frac{\mathrm{e}^{-\alpha} - \mathrm{e}^{-\beta}}{\beta-\alpha}(\alpha - 2\beta) \\
&\ge q\mathrm{e}^{-1/q} - \beta\mathrm{e}^{-\beta} - (\mathrm{e}^{-\alpha} - \mathrm{e}^{-\beta}) - \beta\frac{\mathrm{e}^{-\alpha} - \mathrm{e}^{-\beta}}{\beta-\alpha} \\
&\ge q\mathrm{e}^{-\beta} - \beta\mathrm{e}^{-\beta} - (\mathrm{e}^{-\alpha} - \mathrm{e}^{-\beta}) - \beta\mathrm{e}^{-\beta}\frac{\mathrm{e}^{-(\alpha-\beta)} - 1}{\beta-\alpha} \\
&\ge q\mathrm{e}^{-\beta} - \beta\mathrm{e}^{-\beta} - (\mathrm{e}^{-\alpha} - \mathrm{e}^{-\beta}) - \beta\mathrm{e}^{-\beta}(1 + \frac{1}{2}(\beta-\alpha)) \\
&= (q - 2\beta)\mathrm{e}^{-\beta} - (\mathrm{e}^{-\alpha} - \mathrm{e}^{-\beta}) - (\beta-\alpha)\beta\mathrm{e}^{-\beta}.
\end{aligned}$$

As $\alpha \to 1/q$, we get $\alpha \to \beta$ and so the latter two terms tend to 0. So this allows us to bound $f'(\alpha)$ away from 0 when α is close to the discontinuity at $\alpha = 1/q$. In particular, for $1.489 < q < 1.49$ we have $f'(\alpha) > .01$ when $.64 \le a < 1/q$ and so $f(\alpha)$ has no roots in that range. Having dealt with this discontinuity, it is straightforward to check that for the same range of q, $f(\alpha)$ has no roots in $0 \le \alpha \le .58$, thus proving the lemma.

3. Path coupling and the proof of Theorem 1. Here, we prove Theorem 1. We consider a fixed small ϵ, ρ to be named later. We begin with a burn-in period of $\tau n \log n$ steps, where $\tau > 30$ is as in Lemma 2. All of our analysis will assume that $\tau n \log n < t < n^2$. For now, we assume that for every vertex v and pair of colors c_1, c_2, we have (i) $|L(v)| \ge C\mathrm{e}^{-d(v)/C} - \epsilon\Delta$ and (ii) v has fewer than $\gamma d(v) - \epsilon\Delta$ neighbors w with $d(w) \ge \rho\Delta$ and $c_1, c_2 \notin L(w)$. Later we will account for the $O(n^{-5})$ probability that this is not the case.

We use the path coupling technique of Bubley and Dyer [1]. To do so, we couple two chains $X_0, X_1, \ldots$ and $W_0, W_1, \ldots$ with arbitrary initial colorings X_0, W_0 and show that, with high probability, they coincide within $O(n \log n)$ steps. To prove this fact using path coupling, at any time t, we consider a "path" of possibly improper colorings $X_t = Z_0, Z_1, \ldots, Z_h = W_t$. We define this path as follows. Consider an arbitrary ordering of the vertices $v_1, \ldots, v_n$. To form Z_1 from Z_0, we change the color of the first vertex on which X_t and W_t differ from its color in X_t to its color in W_t.

To form Z_2 from Z_1, we change the color of the second vertex on which X_t, W_t differ, and so on. Thus, h is the Hamming distance between X_t, W_t, i.e., the number of vertices on which they differ. (If $X_t = W_t$ then $h = 0$ and $X_t = Z_0 = W_t$.)

We couple the chains as follows: We carry out a step of $Z_0 = X_t$, thus obtaining $Z_0^{'} = X_{t+1}$. Then we maximally couple a random choice for Z_1 to the choice for Z_0, thus obtaining $Z_1^{'}$. (Recall that, even though Z_r may not be a proper coloring, we can apply a step of our process to it.) Repeatedly, we maximally couple Z_i to Z_{i-1} obtaining $Z_i^{'}$, finally yielding $W_{t+1} = Z_h^{'}$.

Recall that in our key Lemma 2, vertices of degree less than $\rho\Delta$ are not included in the sum $R_{c_1,c_2}(v)$. Because of this, we need to modify the notion of Hamming distance as follows.

Suppose that we are given a particular $\rho > 0$. For any two (not necessarily proper) colorings X, W, we define their *weighted Hamming distance* $H^{'}(X, W)$ to be the number of vertices v with $d(v) > \rho\Delta$ and $X(v) \neq W(v)$ plus 3ρ times the number of vertices v with $d(v) \leq \rho\Delta$ and $X(v) \neq W(v)$. Note that $H^{'}(X_t, W_t) = \sum_{i=0}^{h-1} H^{'}(Z_i, Z_{i+1})$. Note further that $H^{'}(X_{t+1}, W_{t+1}) \leq \sum_{i=0}^{h-1} H^{'}(Z_i^{'}, Z_{i+1}^{'})$, since if $X_{t+1}(v) \neq W_{t+1}(v)$ then $Z_i^{'}(v) \neq Z_{i+1}^{'}(v)$ for at least one i.

We will prove that, after an $O(n \log n)$ burn-in period, the expected value of the change of the weighted Hamming distance between any pair Z_r, Z_{r+1} is at most $-\psi/n$ for some constant $\psi > 0$. Thus, the expected value of the change of the weighted Hamming distance between X, W is at most $-h \times \psi/n < -\psi/n$.

To prove this, we need to know that, with high probability, the bound in Lemma 2 applies to each Z_r. So for each $0 \leq s \leq n$ and step t, we define M_t^s to be the (possibly improper) coloring in which vertices $v_1, \ldots, v_s$ have their color from W_t and $v_{s+1}, \ldots, v_n$ have their colors from X_t. Note that at time t, each Z_r is equal to M_t^s for at least one value of s. At time t, for each vertex v and for each $0 \leq s \leq n$, we define $L^s(v)$ to be the set of colors which do not appear in M_t^s on the neighborhood of v. $L^X(v) = L^0(v)$ is the set of colors which do not appear in X on $N(v)$, and $L^W(v) = L^n(v)$ is the set of colors which do not appear in W on $N(v)$.

LEMMA 8. *For every $\epsilon, \rho > 0$, there exist constants D, τ such that with probability at least $1 - n^{-5}$, for every vertex v, colors c_1, c_2, $0 \leq s \leq n$ and time $\tau n \log n \leq t \leq n^2$, we have the following: Define $\theta = \theta_{c_1,c_2}^s(v)$ to be the set of neighbors w of v with $d(w) \geq \rho\Delta$ and with at least one of c_1, c_2 not appearing in M_t^s on $N(w) - v$, and define*

$$R_{c_1,c_2}^s(v) = \sum_{w \in \theta} \frac{1}{|L(w)|}.$$

Then

$$R_{c_1,c_2}^s(v) \leq \frac{1 - (1 - \mathrm{e}^{-1/q})^2}{q\mathrm{e}^{-1/q}} \times \frac{d(v)}{\Delta} + \epsilon.$$

To prove Lemma 8, we define

$$T_t^s(v, c) = \sum_{v_j \in N(v); j \leq s; c \in L^W_{\tilde{t}(v_j,t)}(v_j)} \frac{1}{|L^W_{\tilde{t}(v_j,t)}(v_j)|} + \sum_{v_j \in N(v); j > s; c \in L^X_{\tilde{t}(v_j,t)}(v_j)} \frac{1}{|L^X_{\tilde{t}(v_j,t)}(v_j)|},$$

and we modify Lemma 5 to the following.

LEMMA 9. *For each $1 \le k \le k^*$, with probability at least $1 - n^{-5}$, for every v with $d(v) > \rho\Delta$, color c, $0 \le s \le n$ and $30kn \log n \le t \le n^2$, we have at time t*

(a) $q\mathrm{e}^{-(1+\zeta)d(v)/C}\Delta \le |L^X(v)|, |L^W(v)| \le \ell_k^{(d(v))}\Delta$;

(b) $a_k d(v)/\Delta \le T_t^s(v,c) \le b_k d(v)/\Delta$.

The proof of part (a) is essentially the same as in Lemma 5. To prove part (b), for each $w_i = v_j$, $\mathcal{H}$ exposes the colors that $w_{i,1,1}, \dots, w_{i,\Delta-1,\Delta-1}$ receive in the chain W if $j \le s$, and exposes the colors they receive in X otherwise. The rest of the proof is the same. The exponent of n in the probability bound changes from -6 to -5 because of the extra n choices for s.

Then we prove Lemma 8 from Lemma 9 by defining $\mathcal{H}$ in the same way.

For now we assume that for every vertex v, $0 \le s \le n$ and pair of colors c_1, c_2, we have

$$R^s_{c_1,c_2}(v) \le \frac{1-(1-\mathrm{e}^{-1/q})^2}{q\mathrm{e}^{-1/q}}\frac{d(v)}{\Delta} + \epsilon.$$

Later we will account for the $O(n^{-5})$ probability that this is not the case.

Now, consider any Z_r and Z_{r+1}. They differ on exactly one vertex, say v which has color c_1 in Z_r and c_2 in Z_{r+1}. We apply one step of our process to Z_r and to Z_{r+1}, coupled as described in the introduction.

Case 1. $d(v) > \rho\Delta$.

The weighted Hamming distance between Z_r and Z_{r+1} decreases by 1 iff we select v. This has probability $1/n$ of occurring. The weighted Hamming distance increases iff we choose a neighbor u of v and assign it color c_1 in Z_r and/or assign it color c_2 in Z_{r+1}. If $d(u) > \rho\Delta$, then it increases by 1; otherwise it increases by 3ρ. Thus, to increase by 1, we must choose a neighbor $u \in \theta$ and so the probability that it increases by 1 is $R_{c_1,c_2}(v)/n$. The probability that it increases by 3ρ is at most $d(v)/(n(C-\Delta))$. Therefore, the expected change in the Hamming distance is at most

$$\frac{1}{n} \times \left(-1 + \frac{1-(1-\mathrm{e}^{-1/q})^2}{q\mathrm{e}^{-1/q} - \epsilon} + \epsilon + \frac{3\rho}{q-1} \right),$$

which is negative if we choose ϵ and ρ to be sufficiently small in terms of q, since we chose q such that $(1-\mathrm{e}^{-1/q})^2 + q\mathrm{e}^{-1/q} > 1$.

Case 2. $d(v) \le \rho\Delta$.

The weighted Hamming distance decreases by 3ρ with probability $\frac{1}{n}$ and it increases with probability at most $\rho\Delta/(n(C-\Delta))$. Since it never increases by more than 1, the expected change in the Hamming distance is at most

$$\frac{1}{n} \times \left(-3\rho + \frac{\rho}{q-1} \right),$$

which is negative for ϵ sufficiently small since $q - 1 > 1/3$ for $q > q^*$.

Thus, in either case, the expected change in the weighted Hamming distance is less than $-\psi/n$ for some $\psi = \psi(q) > 0$. This implies that with sufficiently high probability, the weighted Hamming distance drops to 0 within $O(n \log n)$ steps.

Now we still have to account for the $O(n^{-5})$ chance that for some v, t, s, c_1, c_2, $R^s_{c_1,c_2}(v)$ is too large. Let t^* be the first time at which this occurs. Consider the random variable $H^*(t)$ defined as follows. Until time t^*, $H^*(t) = H'(X_t, W_t)$. After time t^*, if $H^*(t-1) > 0$ then $H^*(t) = H^*(t-1) - 1$ with probability $\frac{1}{n}$ and $H^*(t) = H^*(t-1) + 1$ with probability $\frac{1-\psi}{n}$; if $H^*(t-1) = 0$ then $H^*(t) = 0$. $H^*(t)$ is a simple

random walk with negative drift after time $t = \tau n \log n$, and it is straightforward to verify that with high probability, $H^*(t) = 0$ when $t = O(n \log n)$. Furthermore, with probability $1 - O(n^{-4})$, $H'(X_t, W_t) = H^*(t)$ for each $1 \leq t \leq n^2$. Therefore, with high probability, $H'(X_t, W_t) = 0$ when $t = O(n \log n)$. This is enough to prove Theorem 1. See, for example, [4] for the standard argument.

4. Some final comments. In this section, we note that this coupling argument cannot be used for the case $C = q\Delta$ for any $q < q^*$. To see this, consider any polynomial n^x and any Δ-regular graph G with $\Delta \geq D \log n$ and girth at least $D \log \Delta$ for some sufficiently large D in terms of x.

Note that Lemmas 2 through 9 hold for all $q > 1.489$ (and in fact, if needed, we could show that they hold for even smaller q). The only place where we required $q > q^*$ was in the proof of Theorem 1 in section 3. Furthermore, the upper bound $t \leq n^2$ in their statements can be easily increased to n^x. Therefore, if the Glauber dynamics mixes rapidly, then a "typical" coloring will satisfy that for all v, c, $|L_v|$ is arbitrarily close to $Ce^{-1/q}$ and $T(v, c)$ is arbitrarily close to $1/q$. Thus, such a coloring must exist; call it Ψ.

Suppose that we choose Ψ as our initial coloring. Then for the first n^x steps, with high probability, for all v, c, $|L_v|$ is arbitrarily close to $Ce^{-1/q}$ and $T(v, c)$ is arbitrarily close to $1/q$. This implies that with high probability, for any v, c_1, c_2, the number of neighbors of v which have either c_1 or c_2 in their list is arbitrarily close to $(1-(1-e^{-1/q})^2)\Delta$. So if we couple the coloring arising at any time less than n^x with another coloring which differs in exactly one vertex, then since $q < q^*$, the expected change in their Hamming distance will be positive.

Furthermore, since the graph has large girth, we cannot apply the technique from [5] and [9], where by analyzing the expected total change over a few steps, we were able to get some gain from edges in $N(v)$.

So in an extended abstract of this paper [10], the author raised the following question: *Is there any $q < q^*$ and $D > 0$ such that the Glauber dynamics for $q\Delta$-colorings mixes in polytime on graphs with girth at least D and maximum degree Δ at least $D \log n$?*

He noted that a positive answer would require a substantial new idea. Very recently, Hayes and Vigoda [6] provided such an answer, proving that any $q > 1$ will do even when the girth is as small as 9. Their substantial new idea was to use a "non-Markovian coupling." We refer the reader to their paper for further description.

Acknowledgments. The author is grateful to Martin Dyer and Alan Frieze for some helpful discussions, for providing him with an early copy of their paper [2], and for pointing out an error in an early version. He is also grateful to two anonymous referees for suggestions that greatly improved the presentation.

REFERENCES

[1] R. Bubley and M. Dyer, *Path coupling: A technique for proving rapid mixing in Markov chains*, in Proceedings of the 28th Annual Symposium on Foundations of Computer Science, 1997, pp. 223–231.

[2] M. Dyer and A. Frieze, *Randomly coloring graphs with lower bounds on girth and maximum degree*, Random Structures Algorithms, 23 (2003), pp. 167–179.

[3] M. Dyer, L. Goldberg, C. Greenhill, M. Jerrum, and M. Mitzenmacher, *An extension of path coupling and its application to the Glauber dynamics for path colorings*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, pp. 355–363.

[4] M. Dyer and C. Greenhill, *Random walks on combinatorial objects*, in Surveys in Combinatorics, 1999, J. D. Lamb and D. A. Preece, eds., Cambridge University Press, Cambridge, UK, 1999, pp. 101–136.

[5] M. Dyer, C. Greenhill, and M. Molloy, *Very rapid mixing of the Glauber dynamics for proper colorings on bounded-degree graphs*, Random Structures Algorithms, 20 (2002), pp. 98–114.

[6] T. Hayes and E. Vigoda, *A non-Markovian coupling for sampling colorings*, in Proceedings of FOCS 2003, Boston, MA, 2003.

[7] M. Jerrum, *A very simple algorithm for estimating the number of k-colorings of a low-degree graph*, Random Structures Algorithms, 7 (1995), pp. 157–165.

[8] M. Jerrum, *Mathematical foundations of the Markov chain Monte Carlo method*, in Probabilistic Methods for Algorithmic Discrete Mathematics, M. Habib, C. McDiarmid, J. Ramirez-Alfosin, and B. Reed, eds., Springer, Berlin, 1998, pp. 116–165.

[9] M. Molloy, *Very rapidly mixing Markov Chains for* 2Δ*-colorings and for independent sets in a graph*, Random Structures Algorithms, 18 (2001), pp. 101–115.

[10] M. Molloy, *The Glauber dynamics on the colorings of a graph with large girth and maximum degree*, in Proceedings of STOC 2002, Montreal, QB, Canada, 2002.

[11] J. Salas and A. Sokal, *Absence of phase transition for antiferromagnetic Potts models via the Dobrushin uniqueness theorem*, J. Statist. Phys., 86 (1997), pp. 551–579.

[12] J. van den Berg and J. Steif, *Percolation and the hard-core lattice gas model*, Stochastic Process. Appl., 49 (1994), pp. 179–197.

[13] E. Vigoda, *Improved bounds for sampling colorings*, J. Math. Phys., 41 (2000), pp. 1555–1569.

SIAM J. COMPUT.
Vol. 33, No. 3, pp. 738–760

QUANTUM COMPUTATION AND LATTICE PROBLEMS*

ODED REGEV†

Abstract. We present the first explicit connection between quantum computation and lattice problems. Namely, our main result is a solution to the unique shortest vector problem (SVP) under the assumption that there exists an algorithm that solves the hidden subgroup problem on the dihedral group by coset sampling. Additionally, we present an approach to solving the hidden subgroup problem on the dihedral group by using an average case subset sum routine.

Key words. lattices, quantum computation, shortest vector problem, hidden subgroup problem

AMS subject classifications. 81P68, 68Q25, 68W25, 11H06

DOI. 10.1137/S0097539703440678

1. Introduction.

Quantum computation is a computation model based on quantum physics. Assuming that the laws of nature as we know them are true, this might allow us to build computers that are able to perform tasks that classical computers cannot perform in any reasonable time. One task which quantum algorithms are known to perform much better than classical algorithms is that of factoring large integers. The importance of this problem stems from its ubiquitous use in cryptographic applications. While there are no known polynomial time classical algorithms for this problem, a groundbreaking result of Shor [25] showed a polynomial time quantum algorithm for factoring integers. In the same paper, Shor showed an algorithm for finding the discrete log. However, despite enormous effort, we have only a few other problems for which quantum algorithms provide an exponential speedup (e.g., [12, 5]). Other notable quantum algorithms such as Deutsch and Jozsa's algorithm [6] and Simon's algorithm [26] operate in the black box model. Grover's algorithm [11] provides a square root speedup over classical algorithms.

The current search for new quantum algorithms concentrates on problems which are not known to be NP-hard. These include the graph isomorphism problem and lattice problems. In this paper we are interested in lattice problems or specifically, the unique shortest vector problem (SVP). A lattice is a set of all integral linear combinations of a set of n linearly independent vectors in $\mathbb{R}^n$. This set of n vectors is known as a basis of the lattice. In the SVP we are interested in finding the shortest nonzero vector in a lattice. In the $f(n)$-unique-SVP we are given the additional promise that the shortest vector is shorter by a factor of at least $f(n)$ than all other nonparallel vectors. This problem has also important applications in cryptography. Namely, Ajtai and Dwork's cryptosystem [2] and the recent cryptosystem by Regev [23] are based on the hardness of this lattice problem.

A central problem in quantum computation is the hidden subgroup problem (HSP). Here, we are given a black box that computes a function on elements of a group G. The function is known to be constant and distinct on left cosets of a subgroup $H \leqslant G$, and our goal is to find H. Interestingly, almost all known quantum

*Received by the editors August 27, 2003; accepted for publication (in revised form) February 1, 2004; published electronically April 21, 2004. A preliminary version of this paper appeared in the Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '02). Most of this work was done while the author was at the Institute for Advanced Study, Princeton, NJ, and was supported by Army Research Office grant DAAD19-03-1-0082 and NSF grant CCR-9987845.
http://www.siam.org/journals/sicomp/33-3/44067.html
†EECS Department, University of California, Berkeley, CA 94720 (odedr@cs.berkeley.edu).

algorithms which run superpolynomially faster than classical algorithms solve special cases of the HSP on Abelian groups. Also, it is known that solving the HSP on the symmetric group leads to a solution to graph isomorphism [15]. This motivated research into possible extensions of the HSP to noncommutative groups (see, e.g., [9, 13, 24, 8]). However, prior to this paper the HSP on groups other than the symmetric group and Abelian groups had no known applications.

In this paper we will be interested in the HSP on the dihedral group. The dihedral group of order $2N$, denoted D_N, is the group of symmetries of an N-sided regular polygon. It is isomorphic to the abstract group generated by the element ρ of order n and the element τ of order 2 subject to the relation $\rho\tau = \tau\rho^{-1}$. Although the dihedral group has a much simpler structure than the symmetric group, no efficient solution to the HSP on the dihedral group is known. Ettinger and Høyer [7] showed that one can obtain sufficient statistical *information* about the hidden subgroup with only a polynomial number of queries. However, there is no efficient algorithm that solves the HSP using this information. Currently, the best known algorithm is due to Kuperberg [18] and runs in subexponential time $2^{O(\sqrt{\log N})}$.

The following is the main theorem of this paper. The dihedral coset problem is described in the following paragraph.

THEOREM 1.1. *If there exists a solution to the dihedral coset problem with failure parameter* f*, then there exists a quantum algorithm that solves the* $\Theta(n^{\frac{1}{2}+2\mathsf{f}})$*-unique-SVP.*

The input to the dihedral coset problem (DCP) is a tensor product of a polynomial number of registers. Each register is in the state

$$|0, x\rangle + |1, (x + d) \bmod N\rangle$$

for some arbitrary $x \in \{0, \ldots, N-1\}$, and d is the same for all registers. These can also be thought of as cosets of the subgroup $\{(0,0),(1,d)\}$ in D_N. Our goal is to find the value d. In addition, we say that the DCP has a failure parameter f if each of the registers with probability at most $\frac{1}{(\log N)^{\mathsf{f}}}$ is in the state $|b, x\rangle$ for arbitrary b, x instead of a coset state. We note that any algorithm that solves the dihedral HSP by sampling cosets also solves the DCP for some failure parameter f. The reason is that since the algorithm samples only a polynomial number of cosets, we can take f to be large enough such that with high probability all the registers are coset states. This is summarized in the following corollary.

COROLLARY 1.2. *If there exists a solution to the dihedral HSP that samples cosets (e.g., any solution using the "standard method"), then there exists a quantum algorithm that solves poly*(n)*-unique-SVP.*

The following is the second result of this paper. While still not an efficient solution, it shows a new way to approach the dihedral HSP. In the subset sum problem we are given two integers t, N and a set of numbers. We are asked to find a subset of the numbers that sums to t modulo N. A legal input is an input for which such a subset exists (a formal definition appears in section 4) and we are interested in algorithms that solve a nonnegligible fraction of the inputs.

THEOREM 1.3. *If there exists an algorithm* S *that solves* $\frac{1}{poly(\log N)}$ *of the legal subset sum inputs with parameter* N*, then there exists a solution to the DCP with failure parameter* $\mathsf{f} = 1$*.*

As shown in [7], the dihedral HSP can be reduced to the case where the subgroup is of the form $\{(0,0),(1,d)\}$. Then, by sampling cosets, we obtain states of the form $|0, x\rangle + |1, (x + d) \bmod N\rangle$ with no error. Hence, we have the following.

COROLLARY 1.4. *If there exists an algorithm S that solves $\frac{1}{poly(\log N)}$ of the legal subset sum inputs with parameter N, then there exists a solution to the dihedral HSP.*

Finally, as a curiosity, let us comment that by combining the two previous theorems one can obtain the following corollary.

COROLLARY 1.5. *If there exists an algorithm that solves $\frac{1}{poly(\log N)}$ of the legal subset sum inputs with parameter N, then there exists a quantum algorithm for the $\Theta(n^{2.5})$-unique-SVP.*

This result can be described as a worst-case to average-case quantum reduction. Such reductions are already known in the classical case [1, 3, 4, 20, 23]. The exponent 2.5 in our reduction is better than the one in [1, 3, 4, 20]. However, the reduction in [23], which appeared after the original publication of the current paper, further improves the exponent to 1.5 and hence subsumes our reduction. In addition, unlike the classical reductions, our subset sum problems have a density of one; i.e., the size of the input set is very close to $\log N$. Therefore, some cryptographic applications such as the one by Impagliazzo and Naor [14] cannot be used.

Intuitive overview. Before proceeding to the main part of the paper, we describe our methods in a somewhat intuitive way. First, let us describe the methods used in solving the unique-SVP. Recall that our solution is based on a solution to the DCP. We begin by showing how such a solution can be used to solve a slightly different problem, which we call the two-point problem. Instead of a superposition of two numbers with a fixed difference, our input consists of registers in a superposition of two n-dimensional vectors with a fixed difference. Then the idea is to create an input to the two-point problem in the following way. Start by creating a superposition of many lattice points and collapse the state to just two lattice points whose difference is the shortest vector. Repeating this procedure creates an input to the two-point problem, whose solution is the shortest vector.

Collapsing the state is performed by partitioning the space into cubes. Assume the partition has the property that in each cube there are exactly two lattice points whose difference is the shortest vector. Then we compute the cube in which each point is located and measure the result. The state collapses to a superposition of just the two points inside the cube we measured. The important thing is to make sure that exactly two points are located in each cube. First, in order to make sure that the cubes are not aligned with the lattice, we randomly translate them. The length of the cubes is proportional to the length of the shortest vector. Although the exact length of the shortest vector is unknown, we can try several estimates until we find the right value. Since the lattice has a unique shortest vector, all other nonparallel vectors are considerably longer and do not fit inside a cube. Therefore we know that the difference between any two points inside the same cube is a multiple of the shortest vector. Still, this is not good enough since instead of two points inside each box we are likely to have more points aligned along the shortest vector. Hence, we space out the lattice: instead of creating a superposition of all the lattice points we create a superposition of a subset of the points. The set of points created by this technique has the property that along the direction of the shortest vector there are pairs of points whose difference is the shortest vector and the distance between two such pairs is much larger than the shortest vector. As before, this can be done without knowing the shortest vector by trying several possibilities.

The second part of the paper describes a solution to the DCP with failure parameter 1 which uses a solution to the average-case subset sum problem. Recall that we

are given registers of the form

$$|0, x\rangle + |1, (x + d) \bmod N\rangle,$$

where $x \in \{0, \ldots, N-1\}$ is arbitrary, and we wish to find $d \in \{0, \ldots, N-1\}$. Consider one such register. We begin by applying the Fourier transform to the second part of the register (the one holding x and $x + d$) and then measuring it. If a is the value we measured, the state collapses to a combination of the basis states $|0\rangle$ and $|1\rangle$ such that their phase difference is $2\pi\frac{ad}{N}$. If we were lucky enough to measure $a = 1$, then the phase difference is $2\pi\frac{d}{N}$ and by measuring this phase difference we can obtain an estimation on d. This, however, happens with exponentially small probability. Since the phase is modulo 2π, extracting the value d is much harder when a is larger. Instead, we perform the same process on r registers and let $a_1, \ldots, a_r$ be the values we measure. The resulting tensor state includes a combination of all 2^r different $0, 1$ sequences. The phase of each sequence can be described as follows. By ignoring a fixed phase, we can assume that the phase of the sequence $00\ldots0$ is 0. Then the phase of the sequence $100\ldots0$ is $2\pi\frac{a_1 d}{N}$ and, in general, the phase of the sequence $\alpha_1\alpha_2\ldots\alpha_r$ is $2\pi\frac{d}{N}$ multiplied by the sum of the values a_i for which $\alpha_i = 1$. This indicates that we should try to measure the phase difference of two sequences whose sums differ by 1. However, although we can estimate the phase difference of one qubit, estimating the phase difference of two arbitrary sequences is not possible.

We proceed by choosing r to be very close to $\log N$. This creates a situation in which for almost every $t \in \{0, \ldots, N-1\}$ there is a subset whose sum modulo N is t and, in addition, there are not too many subsets that sum to the same t modulo N. Assume for simplicity that every t has exactly one subset that sums to t modulo N. We calculate for each sequence the value $\lfloor\frac{t}{2}\rfloor$, where t is its sum. After measuring the result, say s, we know that the state is a superposition of two sequences: one that sums to $2s$ and one that sums to $2s + 1$. Notice that since $a_1, \ldots, a_r$ are uniformly chosen between $\{0, \ldots, N-1\}$ we can use them as an input to the subset sum algorithm. The key observation here is that the subset sum algorithm provides the reverse mapping, i.e., from a value t to a subset that sums to t. So, from s we can find the sequence α_1 that sums to $2s$ and the sequence α_2 that sums to $2s + 1$. Since we know that the state is a superposition of $|\alpha_1\rangle$ and $|\alpha_2\rangle$ we can use a unitary transformation that transforms $|\alpha_1\rangle$ to $|0\rangle$ and $|\alpha_2\rangle$ to $|1\rangle$. Now, since the two states differ in one qubit, we can easily measure the phase difference and obtain an estimate on d. This almost completes the description of the DCP algorithm. The estimate on d is only polynomially accurate but in order to find d we need exponential accuracy. Hence, we repeat the same process with pairs whose difference is higher. So, instead of choosing pairs of difference 1 we choose pairs of difference 2 to get an estimate on $2d$, then 4 to get an estimate on $4d$ and so on.[1]

Outline. The next section contains some notation that is used in this paper. The two main sections of this paper are independent. In section 3 we prove Theorem 1.1, and section 4 contains the proof of Theorem 1.3.

2. Preliminaries. We denote the imaginary unit by $\imath$ and use the notation $e(x) = e^{2\pi\imath x}$. Occasionally, we omit the normalization of quantum states. We use the

[1] This description is very similar to the method of exponentially accurate phase estimation used in Kitaev's algorithm [17]. Actually, our case is slightly more difficult because we cannot measure all the multiples 2^i. Nevertheless, we can measure enough multiples of the phase to guarantee exponential accuracy.

term *n-ball* to refer to the n-dimensional solid body and the term *sphere* to refer to its surface. We denote the set $\{1,\ldots,n\}$ by $[n]$. All logarithms are of base 2. We use δ_{ij} to denote the Kronecker delta, i.e., 1 if $i=j$ and 0 otherwise. A sequence $\bar{\alpha}\in\{0,1\}^r$ is identified with the set $\{i \mid \alpha_i = 1\}$. Several constants appear in our proofs. To make it easier to follow, we denote constants with a subscript that is somewhat related to their meaning. Specifically, in section 3, c_{cub} is related to the cubes that partition the space, c_{bal} is related to the radius of the balls, and c_{unq} appears in the guarantee of the unique shortest vector. Also, in section 4 we use c_{r} in the definition of the parameter r, c_{s} in our assumptions on the subset sum subroutine and c_{m} when we prove the existence of matchings.

The following is the formal definition of the DCP.

DEFINITION 2.1. *The input to the DCP with failure parameter* f *consists of* $poly(\log N)$ *registers. Each register is with probability at least* $1-\frac{1}{(\log N)^{\mathsf{f}}}$ *in the state*

$$\frac{1}{\sqrt{2}}(|0,x\rangle + |1,(x+d) \bmod N\rangle)$$

on $1+\lceil\log N\rceil$ *qubits, where* $x\in\{0,\ldots,N-1\}$ *is arbitrary and* d *is fixed. Otherwise, with probability at most* $\frac{1}{(\log N)^{\mathsf{f}}}$*, its state is* $|b,x\rangle$*, where* $b\in\{0,1\}$ *and* $x\in\{0,\ldots,N-1\}$ *are arbitrary. We call such a register a "bad" register. We say that an algorithm solves the DCP if it outputs* d *with probability* $poly(\frac{1}{\log N})$ *in time* $poly(\log N)$*.*

3. A quantum algorithm for unique-SVP. In this section we prove Theorem 1.1. We begin by showing a simple reduction from the two-point problem to the DCP in section 3.1. We then prove a weaker version of Theorem 1.1 with $\Theta(n^{1+2\mathsf{f}})$ instead of $\Theta(n^{\frac{1}{2}+2\mathsf{f}})$ in section 3.2. We complete the proof of Theorem 1.1 in section 3.3. Throughout this section, we use a failure parameter $\mathsf{f}>0$ in order to make our results more general. The reader might find it easier to take $\mathsf{f}=1$.

3.1. The two-point problem.

DEFINITION 3.1. *The input to the two-point problem with failure parameter* f *consists of* $poly(n\log M)$ *registers. Each register is with probability at least* $1-\frac{1}{(n\log(2M))^{\mathsf{f}}}$ *in the state*

$$\frac{1}{\sqrt{2}}(|0,\bar{a}\rangle + |1,\bar{a}'\rangle)$$

on $1+n\lceil\log M\rceil$ *qubits, where* $\bar{a},\bar{a}'\in\{0,\ldots,M-1\}^n$ *are arbitrary such that* $\bar{a}'-\bar{a}$ *is fixed. Otherwise, with probability at most* $\frac{1}{(n\log(2M))^{\mathsf{f}}}$*, its state is* $|b,\bar{a}\rangle$*, where* $b\in\{0,1\}$ *and* $\bar{a}\in\{0,\ldots,M-1\}^n$ *are arbitrary. We say that an algorithm solves the two-point problem if it outputs* $\bar{a}'-\bar{a}$ *with probability* $poly(\frac{1}{n\log M})$ *in time* $poly(n\log M)$*.*

LEMMA 3.2. *If there exists an algorithm that solves the DCP with failure parameter* f*, then there is an algorithm that solves the two-point problem with failure parameter* f*.*

Proof. Consider the following mapping from $\{0,\ldots,M-1\}^n$ to $\{0,\ldots,(2M)^n-1\}$:

$$f(a_1,\ldots,a_n) = a_1 + a_2\cdot 2M + \cdots + a_n(2M)^{n-1}.$$

Given an input to the two-point problem, we create an input to the DCP by using the above mapping on the last $n\lceil\log M\rceil$ qubits of each register. Hence, each register

is with probability at least $1 - \frac{1}{(n(\log 2M))^{\mathrm{f}}}$ in the state

$$\frac{1}{\sqrt{2}}(|0, f(\bar{a})\rangle + |1, f(\bar{a}')\rangle).$$

The difference $f(\bar{a}') - f(\bar{a})$ is

$$(a_1' - a_1) + (a_2' - a_2) \cdot 2M + \cdots + (a_n' - a_n)(2M)^{n-1}$$

and is therefore fixed. Otherwise, with probability at most $\frac{1}{(n(\log 2M))^{\mathrm{f}}}$ the register is in the state $|b, f(\bar{a})\rangle$ for arbitrary $b, \bar{a}$. This is a valid input to the DCP with $N = (2M)^n$ since the probability of a bad register is at most $\frac{1}{(n \log(2M))^{\mathrm{f}}} = \frac{1}{(\log N)^{\mathrm{f}}}$.

Using the DCP algorithm with the above input, we obtain the difference

$$b_1 + b_2 \cdot 2M + \cdots + b_n(2M)^{n-1},$$

where $b_i = a_i' - a_i$. In order to extract the b_i's, we add

$$M + M \cdot 2M + M(2M)^2 + \cdots + M(2M)^{n-1}.$$

Extracting b_i from

$$(b_1 + M) + (b_2 + M) \cdot 2M + \cdots + (b_n + M)(2M)^{n-1}$$

is possible since each $b_i + M$ is an integer in the range 1 to $2M - 1$. The solution to the two-point problem is the vector $(b_1, \ldots, b_n)$. □

3.2. A weaker algorithm. We recall several facts about an LLL-reduced basis. Such a basis can be found for any lattice by using a polynomial time algorithm [19]. Given a basis $\langle \bar{b}_1, \ldots, \bar{b}_n \rangle$, let $\langle \bar{b}_1^*, \ldots, \bar{b}_n^* \rangle$ be its Gram–Schmidt orthogonalization. That is, $\bar{b}_i^*$ is the component of $\bar{b}_i$ orthogonal to the subspace spanned by $\bar{b}_1, \ldots, \bar{b}_{i-1}$. An LLL-reduced basis $\langle \bar{b}_1, \ldots, \bar{b}_n \rangle$ satisfies that

$$\|\bar{b}_i^*\| \leq \sqrt{2}\|\bar{b}_{i+1}^*\|$$

and that for $i > j$,

$$|\langle \bar{b}_i, \bar{b}_j^* \rangle| \leq \frac{1}{2}\|\bar{b}_j^*\|^2.$$

In addition, recall that $\min_i \|\bar{b}_i^*\|$ is a lower bound on the length of the shortest vector. Since $\bar{b}_1^* = \bar{b}_1$ and $\|\bar{b}_1^*\| \leq 2^{(i-1)/2}\|\bar{b}_i^*\|$ we get that the vector $\bar{b}_1$ is at most $2^{(n-1)/2}$ times longer than the shortest vector. Consider the representation of the LLL basis in the orthonormal basis

$$\left\langle \frac{\bar{b}_1^*}{\|\bar{b}_1^*\|}, \ldots, \frac{\bar{b}_n^*}{\|\bar{b}_n^*\|} \right\rangle.$$

The vector $\bar{b}_i$ can be written as $(b_{i1}, b_{i2}, \ldots, b_{ii}, 0, \ldots, 0)$. Notice that $b_{ii} = \|\bar{b}_i^*\|$ and that $|b_{ij}| \leq \frac{1}{2}\|\bar{b}_j^*\|$ for every $i > j$. In the following, $\bar{u}$ denotes the shortest vector.

LEMMA 3.3. *Consider the representation of the shortest vector $\bar{u}$ in the LLL-reduced lattice basis $\bar{u} = \sum_{i=1}^n u_i \bar{b}_i$. Then $|u_i| \leq 2^{2n}$ for $i \in [n]$.*

Proof. Changing to the orthonormal basis,

$$\bar{u} = \sum_{i=1}^{n} u_i \bar{b}_i = \sum_{i=1}^{n} \left(\sum_{j=i}^{n} u_j b_{j,i} \right) \frac{\bar{b}_i^*}{\|\bar{b}_i^*\|}.$$

In addition, we know that $\|\bar{b}_i^*\| \geq 2^{-(i-1)/2} \|\bar{b}_1^*\| \geq 2^{-n} \|\bar{u}\|$. Hence,

$$\left| \sum_{j=i}^{n} u_j b_{j,i} \right| \leq 2^n \|\bar{b}_i^*\|$$

for every $i \in [n]$. By taking $i = n$ we get that $|u_n|$ is at most 2^n. We continue inductively and show that $|u_k| \leq 2^{2n-k}$. Assume that the claim holds for $u_{k+1}, \ldots, u_n$. Then

$$\left| \sum_{j=k+1}^{n} u_j b_{j,k} \right| \leq \frac{1}{2} \left| \sum_{j=k+1}^{n} u_j \right| \|\bar{b}_k^*\| \leq \frac{1}{2} \left(\sum_{j=k+1}^{n} 2^{2n-j} \right) \|\bar{b}_k^*\| \leq \frac{1}{2} \cdot 2^{2n-k} \|\bar{b}_k^*\|.$$

By the triangle inequality,

$$|u_k b_{k,k}| \leq \left| \sum_{j=k+1}^{n} u_j b_{j,k} \right| + \left| \sum_{j=k}^{n} u_j b_{j,k} \right| \leq \left(\frac{1}{2} 2^{2n-k} + 2^n \right) \|\bar{b}_k^*\| \leq 2^{2n-k} \|\bar{b}_k^*\|$$

and the proof is completed. ∎

Let $p > n^{2+2\mathsf{f}}$ be any fixed prime. The following is the main lemma of this section.

LEMMA 3.4. *For any $\mathsf{f} > 0$, if there exists a solution to the two-point problem with failure parameter f, then the following holds. There exists a quantum algorithm that, given a $(c_{\mathsf{unq}} n^{1+2\mathsf{f}})$-unique lattice for some large enough constant $c_{\mathsf{unq}} > 0$ whose shortest vector is $\bar{u} = \sum_{i=1}^{n} u_i \bar{b}_i$, two integers m, i_0 and a number l returns*

$$\left(u_1, \ldots, u_{i_0 - 1}, \frac{u_{i_0} - m}{p}, u_{i_0 + 1}, \ldots, u_n \right),$$

with probability $1/poly(n)$ if the following conditions hold: $\|\bar{u}\| \leq l \leq 2\|\bar{u}\|$, $u_{i_0} \equiv m \pmod{p}$, and $1 \leq m \leq p - 1$.

We first show how this lemma implies Theorem 1.1 with $\Theta(n^{1+2\mathsf{f}})$ by describing the SVP algorithm. According to Lemma 3.2 and the assumption of the theorem, there exists a solution to the two-point problem with failure parameter f. Hence, Lemma 3.4 implies that there exists an algorithm that, given the right values of l, m, i_0, outputs

$$\left(u_1, \ldots, u_{i_0 - 1}, \frac{u_{i_0} - m}{p}, u_{i_0 + 1}, \ldots, u_n \right).$$

The value l is an estimate of the length of the shortest vector $\bar{u}$. Because the LLL algorithm gives a $2^{(n-1)/2}$-approximation to the length of the shortest vector, one of $(n-1)/2$ different values of l is as required. In addition, since $\bar{u}$ is the shortest vector, $\bar{u}/p$ cannot be a lattice vector, and therefore there exists an i_0 such that $u_{i_0} \not\equiv 0 \pmod{p}$. Hence, there are only $O(pn^2)$ possible values for l, m, and i_0. With each of these values the SVP algorithm calls the algorithm of Lemma 3.4 a polynomial

number of times. With high probability in one of these calls the algorithm returns the vector

$$\left(u_1, \ldots, u_{i_0-1}, \frac{u_{i_0} - m}{p}, u_{i_0+1}, \ldots, u_n\right),$$

from which $\bar{u}$ can be extracted. The results of the other calls can be easily discarded because they are either longer lattice vectors or nonlattice vectors.

Proof of Lemma 3.4. We start by applying the LLL algorithm to the unique lattice in order to create a reduced basis. Denote the resulting basis by $\langle \bar{b}_1, \ldots, \bar{b}_n \rangle$. Let $\langle \bar{e}_1, \ldots, \bar{e}_n \rangle$ be the standard orthonormal basis of $\mathbb{R}^n$.

Let $w_1, \ldots, w_n$ be n real values in $[0, 1)$ and let $M = 2^{4n}$. Assume without loss of generality that $i_0 = 1$. The function f is defined as

$$f(t, \bar{a}) = (a_1 p + tm)\bar{b}_1 + \sum_{i=2}^{n} a_i \bar{b}_i,$$

where $t \in \{0, 1\}$ and $\bar{a} = (a_1, \ldots, a_n) \in \mathcal{A} = \{0, \ldots, M - 1\}^n$. It maps the elements of $\{0, 1\} \times \mathcal{A}$ to lattice points. In addition, consider a lattice vector $\bar{v}$ represented in the orthonormal basis $\bar{v} = \sum_{i=1}^{n} v_i \bar{e}_i$. The function g maps $\bar{v}$ to the vector

$$\left(\lfloor v_1/(c_{\mathsf{cub}} n^{\frac{1}{2}+2\mathsf{f}} \cdot l) - w_1 \rfloor, \ldots, \lfloor v_n/(c_{\mathsf{cub}} n^{\frac{1}{2}+2\mathsf{f}} \cdot l) - w_n \rfloor\right)$$

in $\mathbb{Z}^n$, where the constant $c_{\mathsf{cub}} > 0$ will be specified later.

In the following, we describe a routine that creates one register in the input to the two-point problem that hides the difference

$$\left(u_1, \ldots, u_{i_0-1}, \frac{u_{i_0} - m}{p}, u_{i_0+1}, \ldots, u_n\right).$$

We call the routine $poly(n \log M) = poly(n)$ times in order to create a complete input to the two-point problem. We then call the two-point algorithm and output its result. This completes the proof of the lemma since with probability $1/poly(n \log M) = 1/poly(n)$ our output is correct.

The routine starts by choosing $w_1, \ldots, w_n$ uniformly from $[0, 1)$. We create the state

$$\frac{1}{\sqrt{2M^n}} \sum_{t \in \{0,1\}, \bar{a} \in \mathcal{A}} |t, \bar{a}\rangle.$$

Then we compute the function $F = g \circ f$ and measure the result, say $r_1, \ldots, r_n$. The state collapses to (normalization omitted)

$$\sum_{\substack{t \in \{0,1\} \quad \bar{a} \in \mathcal{A} \\ F(t, \bar{a}) = (r_1, \ldots, r_n)}} |t, \bar{a}\rangle |r_1, \ldots, r_n\rangle.$$

This completes the description of the routine. Its correctness is shown in the next two claims.

Claim 3.5. *For every $\bar{r} \in \mathbb{Z}^n$, there is at most one element of the form $(0, \bar{a})$ and at most one element of the form $(1, \bar{a}')$ that get mapped to $\bar{r}$ by F. Moreover, if both $(0, \bar{a})$ and $(1, \bar{a}')$ get mapped to $\bar{r}$, then $\bar{a}' - \bar{a}$ is the vector*

$$\left(\frac{u_1 - m}{p}, u_2, \ldots, u_m\right).$$

Proof. Consider two different lattice points in the image of f, $\bar{v} = f(t, \bar{a})$ and $\bar{v}' = f(t', \bar{a}')$, that get mapped to $\bar{r}$ by g. Let $\bar{v} = \sum_{i=1}^n v_i \bar{e}_i$ and $\bar{v}' = \sum_{i=1}^n v_i' \bar{e}_i$ be their representation in the orthonormal basis. If $\bar{v}' - \bar{v}$ is not a multiple of the shortest vector, then

$$\|\bar{v}' - \bar{v}\| > c_{\mathsf{unq}} n^{1+2\mathsf{f}} \|\bar{u}\| \geq \frac{1}{2} c_{\mathsf{unq}} n^{1+2\mathsf{f}} \cdot l.$$

Therefore, there exists a coordinate $i \in [n]$ such that $|v_i' - v_i| \geq \frac{1}{2} c_{\mathsf{unq}} n^{\frac{1}{2}+2\mathsf{f}} \cdot l$, and for $c_{\mathsf{unq}} > 2c_{\mathsf{cub}}$ this implies $g(\bar{v}) \neq g(\bar{v}')$ no matter how $w_1, \ldots, w_n$ are chosen. Hence, $\bar{v}' - \bar{v} = k \cdot \bar{u}$ for some integer $k \neq 0$. By considering the first coordinate of $\bar{v}' - \bar{v}$ in the lattice basis, we get that

$$(a_1' p + t' m) - (a_1 p + t m) \equiv k \cdot m \pmod{p}.$$

This implies that $k \equiv t' - t \pmod{p}$. If $t = t'$, then $k \equiv 0 \pmod{p}$, which implies that $|k| \geq p$. Thus,

$$\|\bar{v}' - \bar{v}\| \geq p \|\bar{u}\| > c_{\mathsf{cub}} n^{1+2\mathsf{f}} \cdot l$$

and, again, $g(\bar{v}) \neq g(\bar{v}')$. This proves the first part of the claim. For the second part, let $t = 0$ and $t' = 1$. Then $k \equiv 1 \pmod{p}$. As before, this can happen only when $k = 1$, and hence the second part of the claim holds. □

Hence, it is enough to show that the probability that this register is bad is low enough. The probability of measuring $|r_1, \ldots, r_n\rangle$ equals

$$\frac{1}{2M^n} \cdot \left| \{(t, \bar{a}) \mid F(t, \bar{a}) = (r_1, \ldots, r_n)\} \right|.$$

Notice that this probability is the same as the probability that $F(t, \bar{a}) = (r_1, \ldots, r_n)$ for randomly chosen t and $\bar{a}$. Hence, we consider a randomly chosen t and $\bar{a}$. If $t = 0$, let

$$\bar{a}' = \left(a_1 + \frac{u_1 - m}{p}, a_2 + u_2, \ldots, a_n + u_n \right),$$

and if $t = 1$ let

$$\bar{a}' = \left(a_1 - \frac{u_1 - m}{p}, a_2 - u_2, \ldots, a_n - u_n \right).$$

Claim 3.6. *With probability at least* $1 - \frac{1}{(n \log(2M))^{\mathsf{f}}}$, *for randomly chosen* t *and* $\bar{a}$, $\bar{a}'$ *is in* $\mathcal{A}$ *and* $F(1 - t, \bar{a}') = F(t, \bar{a})$.

Proof. We assume that $t = 0$, and the proof for $t = 1$ is similar. According to Lemma 3.3, $|u_i| < 2^{2n}$. Hence, unless there exists an i for which $a_i < 2^{2n}$ or $a_i > M - 2^{2n}$, $\bar{a}'$ is guaranteed to be in $\mathcal{A}$. This happens with probability at most $n 2^{2n+1}/M$ because $\bar{a}$ is a random element of $\mathcal{A}$.

Notice that $f(1, \bar{a}') - f(0, \bar{a}) = \bar{u}$. Since $w_1, \ldots, w_n$ are randomly chosen, the probability that $F(1 - t, \bar{a}')$ and $F(t, \bar{a})$ differ on the ith coordinate is at most

$$\frac{|\langle \bar{u}, \bar{e}_i \rangle|}{c_{\mathsf{cub}} n^{\frac{1}{2}+2\mathsf{f}} \cdot l} \leq \frac{|\langle \bar{u}, \bar{e}_i \rangle|}{c_{\mathsf{cub}} n^{\frac{1}{2}+2\mathsf{f}} \cdot \|\bar{u}\|}.$$

By the union bound, the probability that $F(1-t,\bar{a}') \neq F(t,\bar{a})$ is at most

$$\frac{\sum_i |\langle \bar{u}, \bar{e}_i \rangle|}{c_{\mathsf{cub}} n^{\frac{1}{2}+2\mathsf{f}} \cdot \|\bar{u}\|} \leq \frac{1}{c_{\mathsf{cub}} n^{2\mathsf{f}}},$$

where we used the fact that the l_1 norm of a vector is at most $\sqrt{n}$ times its l_2 norm.

The sum of the two error probabilities $n\frac{2^{2n+1}}{M} + \frac{1}{c_{\mathsf{cub}} n^{2\mathsf{f}}}$ is at most $\frac{1}{(n\log(2M))^{\mathsf{f}}}$ for c_{cub} large enough. □

This concludes the proof of Lemma 3.4. □

3.3. An improved algorithm. In this section we complete the proof of Theorem 1.1. The algorithm we describe has many similarities with the one in the previous section. The main difference is that it is based on n-dimensional balls instead of cubes. The idea is to construct a ball of the right radius around lattice points and to show that if two lattice points are close, then the two balls have a large intersection, while for any two far lattice points the balls do not intersect. For technical reasons, we will assume in this section that the lattice is a subset of $\mathbb{Z}^n$. Any lattice with rational points can be scaled so that it is a subset of $\mathbb{Z}^n$. We begin with some technical claims.

CLAIM 3.7. *For any $R > 0$, let B_n be the ball of radius R centered around the origin in $\mathbb{R}^n$ and let $B'_n = B_n + \bar{d}$ for some vector $\bar{d}$ be a shifted ball. Then the relative n-dimensional volume of their intersection is at least $1 - O(\sqrt{n}\|\bar{d}\|/R)$, i.e.,*

$$\frac{\mathrm{vol}(B_n \cap B'_n)}{\mathrm{vol}(B_n)} \geq 1 - O(\sqrt{n}\|\bar{d}\|/R).$$

Proof. Consider a point $\bar{x} \in \mathbb{R}^n$ such that $\langle \bar{x}, \bar{d} \rangle / \|\bar{d}\| \geq \|\bar{d}\|/2$, i.e., a point which is closer to the center of B'_n than to the center of B_n. Notice that $\bar{x} \in B_n$ implies $\bar{x} \in B'_n$. In other words, the cap C_n of B_n given by all such points $\bar{x}$ is contained in $B_n \cap B'_n$. By using a symmetric argument for points $\bar{x} \in \mathbb{R}^n$ such that $\langle \bar{x}, \bar{d} \rangle / \|\bar{d}\| < \|\bar{d}\|/2$, we get

$$\mathrm{vol}(B_n \cap B'_n) = 2 \cdot \mathrm{vol}(C_n).$$

We can lower bound the volume of C_n by half the volume of B_n minus the volume of an n-dimensional cylinder of radius R and height $\|\bar{d}\|/2$:

$$\mathrm{vol}(C_n) \geq \frac{1}{2}\mathrm{vol}(B_n) - \frac{\|\bar{d}\|}{2}\mathrm{vol}(B_{n-1}),$$

where B_{n-1} is the $n-1$-ball of radius R. We complete the proof by using the estimate $\mathrm{vol}(B_{n-1})/\mathrm{vol}(B_n) = O(\sqrt{n}/R)$,

$$\mathrm{vol}(C_n)/\mathrm{vol}(B_n) \geq \frac{1}{2} - O(\sqrt{n}\|\bar{d}\|/R). \qquad \square$$

In the algorithm we will actually represent the balls using points of a fine grid. Therefore, we would like to say that the above claim still holds if we consider the number of grid points inside B_n, B'_n, and $B_n \cap B'_n$ instead of their volumes. The following claim is more than enough for our needs.

CLAIM 3.8 (special case of Proposition 8.7 in [21]). *Let L be an integer and consider the scaled integer grid $\frac{1}{L}\mathbb{Z}^n$. Then, for any convex body Q that contains a ball of radius $r \geq \frac{1}{L}n^{1.5}$,*

$$\left| \frac{|\frac{1}{L}\mathbb{Z}^n \cap Q|}{L^n \mathrm{vol}(Q)} - 1 \right| < \frac{2n^{1.5}}{rL}.$$

COROLLARY 3.9. *Let $L = 2^n$ and consider the scaled integer grid $\frac{1}{L}\mathbb{Z}^n$. For any $R \geq 1$, let B_n be the ball of radius R centered around the origin in $\mathbb{R}^n$ and let $B'_n = B_n + \bar{d}$ for some vector $\bar{d}$ such that $R/poly(n) \leq \|\bar{d}\| \leq R$. Then the relative number of grid points in their intersection is at least $1 - O(\sqrt{n}\|\bar{d}\|/R)$, i.e.,*

$$\frac{|\frac{1}{L}\mathbb{Z}^n \cap B_n \cap B'_n|}{|\frac{1}{L}\mathbb{Z}^n \cap B_n|} \geq 1 - O(\sqrt{n}\|\bar{d}\|/R).$$

Proof. We first note that B_n, B'_n, and $B_n \cap B'_n$ all contain the ball of radius $R/2 \geq 1/2$ centered around $\bar{d}/2$. Using Claim 3.8 we obtain that the number of grid points in these bodies approximates their volume up to a multiplicative error of $\frac{2n^{1.5}}{L/2} = 2^{-\Omega(n)}$. We complete the proof by using Claim 3.7. ∎

Let $D(\cdot,\cdot)$ denote the trace distance between two quantum states [22], i.e.,

$$D(\sigma_1, \sigma_2) = \frac{1}{2}\mathrm{tr}\sqrt{(\sigma_1 - \sigma_2)^\dagger(\sigma_1 - \sigma_2)}.$$

It is known that the trace distance represents the maximum probability of distinguishing between the two states using quantum measurements. We need the following simple bound on the trace distance.

CLAIM 3.10. *For all $k > 0$ and density matrices $\sigma_1, \ldots, \sigma_k, \sigma'_1, \ldots, \sigma'_k$,*

$$D(\sigma_1 \otimes \cdots \otimes \sigma_k, \sigma'_1 \otimes \cdots \otimes \sigma'_k) \leq \sum_{i=1}^{k} D(\sigma_i, \sigma'_i).$$

Proof. Using the triangle inequality,

$$\begin{aligned} D(\sigma_1 \otimes \cdots \otimes \sigma_k,\ & \sigma'_1 \otimes \cdots \otimes \sigma'_k) \\ &\leq D(\sigma_1 \otimes \cdots \otimes \sigma_k,\ \sigma'_1 \otimes \sigma_2 \otimes \cdots \otimes \sigma_k) \\ &\quad + D(\sigma'_1 \otimes \sigma_2 \otimes \cdots \otimes \sigma_k,\ \sigma'_1 \otimes \sigma'_2 \otimes \sigma_3 \otimes \cdots \otimes \sigma_k) + \cdots \\ &\quad + D(\sigma'_1 \otimes \cdots \otimes \sigma'_{k-1} \otimes \sigma_k,\ \sigma'_1 \otimes \cdots \otimes \sigma'_k) \\ &= D(\sigma_1,\ \sigma'_1) + D(\sigma_2,\ \sigma'_2) + \cdots + D(\sigma_k,\ \sigma'_k). \end{aligned}$$ ∎

In addition, we will need the following lemma.

LEMMA 3.11. *For any $1 \leq R \leq 2^{poly(n)}$, let*

$$|\eta\rangle = \frac{1}{\sqrt{|\frac{1}{L}\mathbb{Z}^n \cap B_n|}} \sum_{\bar{x} \in \frac{1}{L}\mathbb{Z}^n \cap B_n} |\bar{x}\rangle$$

be the uniform superposition on grid points inside a ball of radius R around the origin where $L = 2^n$. Then, for any $c > 0$, a state $|\tilde{\eta}\rangle$ whose trace distance from $|\eta\rangle$ is at most $1/n^c$ can be efficiently computed.

Proof. In order to bound the trace distance, we will use the fact that for any two pure states $|\psi_1\rangle, |\psi_2\rangle$,

$$D(|\psi_1\rangle, |\psi_2\rangle) = \sqrt{1 - |\langle\psi_1|\psi_2\rangle|^2} \leq \||\psi_1\rangle - |\psi_2\rangle\|_2. \tag{3.1}$$

The first equality appears in [22], and the inequality follows by a simple calculation.

Consider the (continuous) uniform probability distribution q over B_n. Then one can define its discretization q' to the grid $\frac{1}{L}\mathbb{Z}^n$ as

$$q'(\bar{x}) = \int_{\bar{x}+[0,1/L]^n} q(\bar{y})d\bar{y}$$

for $\bar{x} \in \frac{1}{L}\mathbb{Z}^n$. In other words, $q'(\bar{x})$ is proportional to the volume of the intersection of B_n with the cube $\bar{x} + [0, 1/L]^n$. Notice that for points $\bar{x}$ such that $\bar{x} + [0, 1/L]^n$ is completely contained in B_n, $q'(\bar{x}) = 1/(L^n \mathrm{vol}(B_n))$. We claim that the state

$$|\eta'\rangle = \sum_{\bar{x} \in \frac{1}{L}\mathbb{Z}^n} \sqrt{q'(\bar{x})}|\bar{x}\rangle$$

is exponentially close to $|\eta\rangle$. Intuitively, this holds since the two differ only on points which are very close to the boundary of the ball, namely, of distance $\sqrt{n}/L$ from the boundary. The number of such points is negligible compared to the number of points in the interior of the ball. More formally, define

$$|\eta''\rangle = \sqrt{\frac{L^n \mathrm{vol}(B_n)}{|\frac{1}{L}\mathbb{Z}^n \cap B_n|}}|\eta'\rangle.$$

Using (3.1),

$$D(|\eta\rangle, |\eta'\rangle) \leq \||\eta'\rangle - |\eta\rangle\|_2 \leq \||\eta'\rangle - |\eta''\rangle\|_2 + \||\eta''\rangle - |\eta\rangle\|_2.$$

The first term is at most $2^{-\Omega(n)}$ according to Claim 3.8. For the second term, notice that the amplitudes of $|\eta''\rangle$ and $|\eta\rangle$ are the same except possibly on points $\bar{x}$ of distance $\sqrt{n}/L$ from the boundary. Using Claim 3.8 again we get that the fraction of such points is closely approximated by one minus the ratio of volumes of the ball of radius $R - \sqrt{n}/L$ and the ball of radius R. This ratio of volumes is

$$(1 - \sqrt{n}/(RL))^n \geq (1 - \sqrt{n}/L)^n \geq 1 - n^{1.5}/L = 1 - 2^{-\Omega(n)}.$$

In the following we show how to approximate the state $|\eta'\rangle$. This idea is essentially due to Grover and Rudolph [10]. Let $m \in \mathbb{Z}$ be large enough so that B_n is contained in the cube $[-2^m, 2^m]^n$. Using our assumption on R, $m < n^{c_1}$ for some $c_1 \geq 1$. We represent $\bar{x}$ using $K = n(m + 1 + \log L) < 2n^{1+c_1}$ qubits, i.e., a block of $m + 1 + \log L$ qubits for each dimension. Hence, we can write $|\eta'\rangle$ as

$$|\eta'\rangle = \sum_{x_1,\ldots,x_K \in \{0,1\}} \sqrt{q'(x_1, \ldots, x_K)}|x_1, \ldots, x_K\rangle.$$

We now show an equivalent way of writing $|\eta'\rangle$. Let us extend the definition of q' in the following way: for any $k \leq K$ and any $x_1, \ldots, x_k \in \{0, 1\}$ define $q'(x_1, \ldots, x_k)$ as the sum of $q'(x_1, \ldots, x_k, x_{k+1}, \ldots, x_K)$ over all sequences $x_{k+1}, \ldots, x_K \in \{0, 1\}$. Notice that $q'(x_1, \ldots, x_k)$ corresponds to the volume of the intersection of B_n with a certain cuboid (also known as a rectangular parallelepiped). For example, $q'(0) = q'(1) = \frac{1}{2}$ since they represent the intersection of B_n with two halves of the cube $[-2^m, 2^m]^n$. Using the definition $s(x_1) = q'(x_1)$ and for $k > 1$, $s(x_1, \ldots, x_k) = q'(x_1, \ldots, x_k)/q'(x_1, \ldots, x_{k-1})$, we see that

$$|\eta'\rangle = \sum_{x_1 \in \{0,1\}} \sqrt{s(x_1)} \sum_{x_2 \in \{0,1\}} \sqrt{s(x_1, x_2)} \ldots \sum_{x_K \in \{0,1\}} \sqrt{s(x_1, \ldots, x_K)}|x_1, \ldots, x_K\rangle.$$

The algorithm starts with all K qubits in the state $|0\rangle$ and sets one qubit at a time. The first qubit is rotated to the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Assume we are now in the kth step after setting the state of qubits $1, \ldots, k-1$. We use the fact that there

exists a classical algorithm for approximating the volume of a convex body up to any $1/poly(n)$ error (see [16] and the references therein). The body should be provided by a "well-guaranteed weak membership oracle," i.e., a sphere containing the body, a sphere contained in the body, both of nonzero radius, and an oracle that, given a point, decides whether it is inside the body or not. It is easy to construct such two spheres and an oracle for a body given by the intersection of a ball with a cuboid. Hence, we can compute two values $\tilde{s}(x_1,\ldots,x_{k-1},0)$ and $\tilde{s}(x_1,\ldots,x_{k-1},1)$ such that

$$\tilde{s}(x_1,\ldots,x_{k-1},0)+\tilde{s}(x_1,\ldots,x_{k-1},1)=1$$

and

$$\left|\frac{\tilde{s}(x_1,\ldots,x_{k-1},i)}{s(x_1,\ldots,x_{k-1},i)}-1\right|<n^{-c_2}$$

for $i=0,1$ and some constant c_2 which will be chosen later. Then we rotate the ith qubit to the state

$$\sqrt{\tilde{s}(x_1,\ldots,x_{k-1},0)}|0\rangle+\sqrt{\tilde{s}(x_1,\ldots,x_{k-1},1)}|1\rangle.$$

This completes the description of the procedure.

Notice that the amplitude of each basis state $|x_1,\ldots,x_K\rangle$ in the resulting state $|\tilde{\eta}\rangle$ is given by

$$\prod_{k=1}^{K}\sqrt{\tilde{s}(x_1,\ldots,x_k)}\geq(1-n^{-c_2})^K\prod_{k=1}^{K}\sqrt{s(x_1,\ldots,x_k)}.$$

Hence the inner product $\langle\tilde{\eta}|\eta'\rangle$ is at least

$$\begin{aligned}(1-n^{-c_2})^K&\sum_{x_1,\ldots,x_K\in\{0,1\}}\prod_{k=1}^{K}s(x_1,\ldots,x_k)\\&=(1-n^{-c_2})^K\sum_{x_1,\ldots,x_K\in\{0,1\}}q'(x_1,\ldots,x_K)\\&=(1-n^{-c_2})^K\geq 1-K\cdot n^{-c_2}\geq 1-2n^{1+c_1-c_2}.\end{aligned}$$

Using (3.1),

$$D(|\eta'\rangle,|\tilde{\eta}\rangle)=\sqrt{1-|\langle\tilde{\eta}|\eta'\rangle|^2}<n^{-c}$$

for a large enough c_2. □

Let $p>n^{2+2\mathsf{f}}$ be any fixed prime. The following is the main lemma of this section. It essentially replaces Lemma 3.4 and hence implies Theorem 1.1.

LEMMA 3.12. *For any $\mathsf{f}>0$, if there exists a solution to the two-point problem with failure parameter f, then the following holds. There exists a quantum algorithm that, given a $(c_{\mathsf{unq}}n^{\frac{1}{2}+2\mathsf{f}})$-unique lattice for some large enough constant $c_{\mathsf{unq}}>0$ whose shortest vector is $\bar{u}=\sum_{i=1}^{n}u_i\bar{b}_i$, two integers m,i_0 and a number l returns*

$$\left(u_1,\ldots,u_{i_0-1},\frac{u_{i_0}-m}{p},u_{i_0+1},\ldots,u_n\right)$$

with probability $1/poly(n)$ if the following conditions hold: $\|\bar{u}\|\leq l\leq 2\|\bar{u}\|$, $u_{i_0}\equiv m \pmod p$, and $1\leq m\leq p-1$.

Proof. As before, let $\langle \bar{b}_1, \ldots, \bar{b}_n \rangle$ be an LLL-reduced basis, let $M = 2^{4n}$, and assume that $i_0 = 1$. We also define $f(t, \bar{a})$ as before. Assume that the number of registers needed by the two-point algorithm is at most n^{c_1} for some constant $c_1 > 0$.

The algorithm starts by calling the routine of Lemma 3.11 n^{c_1} times with accuracy parameter n^{-c_2} and $R = c_{\mathsf{bal}} n^{\frac{1}{2}+2\mathsf{f}} \cdot l$ for some constants $c_2, c_{\mathsf{bal}} > 0$. The state we obtain is

$$|\tilde{\eta}_1\rangle \otimes \cdots \otimes |\tilde{\eta}_{n^{c_1}}\rangle, \tag{3.2}$$

where each $|\tilde{\eta}_i\rangle$ has a trace distance of at most n^{-c_2} from $|\eta\rangle$. According to Claim 3.10, the above tensor product has a trace distance of at most $n^{c_1-c_2}$ from $|\eta\rangle^{\otimes n^{c_1}}$. In the following we show that the algorithm succeeds with probability at least n^{-c_3} for some $c_3 > 0$ given the state $|\eta\rangle^{\otimes n^{c_1}}$. This would complete the proof since, given the state in (3.2), the algorithm succeeds with probability at least $n^{-c_3} - n^{c_1-c_2} > \frac{1}{2} n^{-c_3}$ for large enough c_2.

We describe a routine that given the state $|\eta\rangle$ creates one register in the input to the two-point problem. In order to produce a complete input to the two-point problem, the algorithm calls this routine n^{c_1} times, each time with a new $|\eta\rangle$ register. It then calls the two-point algorithm and outputs the result. As required, the success probability is $1/poly(n \log M) = n^{-c_3}$ for some $c_3 > 0$.

Given $|\eta\rangle$, the routine creates the state

$$\frac{1}{\sqrt{2M^n}} \sum_{t \in \{0,1\}, \bar{a} \in \mathcal{A}} |t, \bar{a}\rangle \otimes |\eta\rangle$$

or, equivalently,

$$\sum_{t \in \{0,1\}, \bar{a} \in \mathcal{A}, \bar{x} \in \frac{1}{L}\mathbb{Z}^n \cap B_n} |t, \bar{a}, \bar{x}\rangle,$$

where B_n is the ball of radius R around the origin and $L = 2^n$. We add the value $f(t, \bar{a})$ to the last register,

$$\sum_{t \in \{0,1\}, \bar{a} \in \mathcal{A}, \bar{x} \in \frac{1}{L}\mathbb{Z}^n \cap B_n} |t, \bar{a}, f(t, \bar{a}) + \bar{x}\rangle.$$

Finally, we measure the last register, and if $\bar{x}'$ denotes the result, the state collapses to

$$\sum_{t \in \{0,1\}, \bar{a} \in \mathcal{A} | \bar{x}' \in f(t,\bar{a}) + \frac{1}{L}\mathbb{Z}^n \cap B_n} |t, \bar{a}, \bar{x}'\rangle.$$

CLAIM 3.13. *For every $\bar{x}'$, there is at most one element of the form $(0, \bar{a})$ and at most one element of the form $(1, \bar{a}')$ such that $\bar{x}' \in f(t, \bar{a}) + \frac{1}{L}\mathbb{Z}^n \cap B_n$. Moreover, if there are two such elements $(0, \bar{a})$ and $(1, \bar{a}')$, then $\bar{a}' - \bar{a}$ is the vector*

$$\left(\frac{u_1 - m}{p}, u_2, \ldots, u_m \right).$$

Proof. Consider two different lattice points in the image of f, $\bar{v} = f(t, \bar{a})$ and $\bar{v}' = f(t', \bar{a}')$, such that $\bar{x}'$ is in both $\bar{v} + \frac{1}{L}\mathbb{Z}^n \cap B_n$ and $\bar{v}' + \frac{1}{L}\mathbb{Z}^n \cap B_n$. This implies that

$$\|\bar{v} - \bar{v}'\| \leq c_{\mathsf{bal}} n^{\frac{1}{2}+2\mathsf{f}} \cdot l \leq 2 c_{\mathsf{bal}} n^{\frac{1}{2}+2\mathsf{f}} \cdot \|\bar{u}\|.$$

For $c_{\mathsf{unq}} > 2c_{\mathsf{bal}}$ this means that $\bar{v}' - \bar{v} = k \cdot \bar{u}$ for some integer $k \neq 0$. As before, by considering the first coordinate of $\bar{v}' - \bar{v}$ in the lattice basis, we get that

$$(a_1' p + t' m) - (a_1 p + t m) \equiv k \cdot m \pmod{p}.$$

Hence, $k \equiv t' - t \pmod{p}$. If $t = t'$, then $k \equiv 0 \pmod{p}$ and therefore $|k| \geq p$, which contradicts the above upper bound on the distance between $\bar{v}$ and $\bar{v}'$. This proves the first part of the claim. For the second part, let $t = 0$ and $t' = 1$. Then $k \equiv 1 \pmod{p}$. As before, this can happen only when $k = 1$, and hence the second part of the claim holds. □

Notice that the probability of measuring $\bar{x}'$ is the same as that obtained by first choosing random t and $\bar{a}$ and then choosing a random point in $f(t, \bar{a}) + \frac{1}{L}\mathbb{Z}^n \cap B_n$. Let us define for any t and $\bar{a}$ the vector $\bar{a}'$ as before.

Claim 3.14. *With probability at least* $1 - \frac{1}{(n \log(2M))^{\mathsf{f}}}$, *for randomly chosen* t *and* $\bar{a}$ *and a random point* $\bar{x}'$ *in* $f(t, \bar{a}) + \frac{1}{L}\mathbb{Z}^n \cap B_n$, $\bar{a}'$ *is in* $\mathcal{A}$ *and* $\bar{x}'$ *is also in* $f(1 - t, \bar{a}') + \frac{1}{L}\mathbb{Z}^n \cap B_n$.

Proof. According to Lemma 3.3, $|u_i| < 2^{2n}$. Hence, unless there exists an i for which $a_i < 2^{2n}$ or $a_i > M - 2^{2n}$, $\bar{a}'$ is guaranteed to be in $\mathcal{A}$. This happens with probability at most $n2^{2n+1}/M$ because $\bar{a}$ is a random element of $\mathcal{A}$.

Fix $\bar{a}, \bar{a}' \in \mathcal{A}$. We would like to show that if $\bar{x}'$ is chosen uniformly from $f(t, \bar{a}) + \frac{1}{L}\mathbb{Z}^n \cap B_n$, then with high probability it is also in

$$f(1 - t, \bar{a}') + \frac{1}{L}\mathbb{Z}^n \cap B_n.$$

By translating both sets by $-f(t, \bar{a})$, we get the equivalent statement that if $\bar{x}'$ is chosen uniformly from $\frac{1}{L}\mathbb{Z}^n \cap B_n$, then with high probability it is also in $(f(1 - t, \bar{a}') - f(t, \bar{a})) + \frac{1}{L}\mathbb{Z}^n \cap B_n$. Since we assumed that our lattice is a subset of $\mathbb{Z}^n$, $f(1 - t, \bar{a}') - f(t, \bar{a}) \in \mathbb{Z}^n$ and the latter set equals $\frac{1}{L}\mathbb{Z}^n \cap (f(1 - t, \bar{a}') - f(t, \bar{a}) + B_n)$. Using Corollary 3.9 and the fact that $\|f(1 - t, \bar{a}') - f(t, \bar{a})\| = \|\bar{u}\| \leq l$, we get that the required probability is at least

$$1 - O(\sqrt{n} l / R) = 1 - O(\sqrt{n} l / (c_{\mathsf{bal}} n^{\frac{1}{2} + 2\mathsf{f}} \cdot l)) = 1 - O(1/(c_{\mathsf{bal}} n^{2\mathsf{f}})).$$

The sum of the two error probabilities $n2^{2n+1}/M + O(1/(c_{\mathsf{bal}} n^{2\mathsf{f}}))$ is at most $\frac{1}{(n \log(2M))^{\mathsf{f}}}$ for c_{bal} large enough. □

This concludes the proof of Lemma 3.12. □

4. The dihedral coset problem. We begin this section with a description of the average-case subset sum problem. We describe our assumptions on the subroutine that solves it and prove some properties of such a subroutine. In the second subsection we present an algorithm that solves the DCP with calls to an average-case subset sum subroutine.

4.1. Subset sum. The subset sum problem is defined as follows. An input is a sequence of numbers $A = (a_1, \ldots, a_r)$ and two numbers t, N. The output is a subset $B \subseteq [r]$ such that $\sum_{i \in B} a_i \equiv t \pmod{N}$. Let a legal input be an input for which there exists a subset B with $\sum_{i \in B} a_i \equiv t \pmod{N}$. For a constant $c_{\mathsf{r}} > 0$, we fix r to be $\log N + c_{\mathsf{r}}$ since we will only be interested in such instances. First we show that there are many legal inputs.

Lemma 4.1. *Let* c_{r} *be a large enough constant. Then, for randomly chosen* $a_1, \ldots, a_r, t$ *in* $\{0, \ldots, N - 1\}$, *the probability that there is no* $B \subseteq [r]$ *such that* $\sum_{i \in B} a_i \equiv t \pmod{N}$ *is at most* $\frac{1}{2}$.

Proof. Fix a value of t. For each $\bar{b} \in \{0,1\}^r, \bar{b} \neq 0^r$, define a random variable $X_{\bar{b}}$ as $\sum_i b_i a_i \bmod N$. It is easy to check that for any $\bar{b} \neq 0^r$, $X_{\bar{b}}$ is uniformly distributed on $\{0,\ldots,N-1\}$ and that the random variables $X_{\bar{b}}$ are pairwise independent. For every $\bar{b} \in \{0,1\}^r, \bar{b} \neq 0^r$, define a random variable $Y_{\bar{b}}$ as 1 if $X_{\bar{b}} = t$ and 0 otherwise. Then the expectation of $Y_{\bar{b}}$ is $\frac{1}{N}$ and its variance is $\frac{1}{N} - \frac{1}{N^2} < \frac{1}{N}$. Hence,

$$E\left[\sum_{\bar{b}} Y_{\bar{b}}\right] = \sum_{\bar{b}} E[Y_{\bar{b}}] = \frac{2^r - 1}{N}.$$

The $Y_{\bar{b}}$'s are defined as a function of the $X_{\bar{b}}$'s and are therefore also pairwise independent. Therefore, by the Chebyshev bound,

$$\Pr\left[\sum_{\bar{b}} Y_{\bar{b}} < \frac{1}{2} \cdot \frac{2^r - 1}{N}\right] \leq 4 \cdot \frac{N}{2^r - 1} \leq \frac{8}{2^{c_r}}.$$

In particular, the probability of $\sum_{\bar{b}} Y_{\bar{b}} = 0$, that is, the probability that there is no B such that $\sum_{i \in B} a_i \equiv t \pmod N$, is at most $\frac{8}{2^{c_r}} = \frac{1}{2}$ for $c_r = 4$. □

We assume that we are given a deterministic subroutine S that answers a $\frac{1}{\log^{c_s} N}$ fraction of the legal subset sum inputs with parameter N, where $c_s > 0$ is any constant.[2] The previous lemma implies that S answers a nonnegligible fraction of all inputs (and not just the legal inputs). We denote by $S(A,t)$ the result of the subroutine S on the input $A = (a_1,\ldots,a_r), t$ and we omit N. This result can be either a set or an error. We assume that whenever $S(A,t)$ is not an error, it is correct; i.e., it represents a subset of A that sums to t modulo N. This can be assumed without loss of generality since we can easily check the correctness of any output of S. Let $S(A)$ denote the set of t's for which the subroutine returns a set and not an error, i.e., $S(A) = \{t \mid S(A,t) \neq error\}$.

LEMMA 4.2. *For randomly chosen $a_1,\ldots,a_r$ in $\{0,\ldots,N-1\}$,*

$$\Pr_A\left[|S(A)| \geq \frac{N}{4\log^{c_s} N}\right] = \Omega\left(\frac{1}{\log^{c_s} N}\right),$$

where $A = (a_1,\ldots,a_r)$.

Proof. Since $S(A,t) \neq error$ only when (A,t) is a legal input,

$$\begin{aligned}
&\Pr_{A,t}[S(A,t) \neq error] \\
&= \Pr_{A,t}[\ S(A,t) \neq error \ \wedge\ (A,t) \text{ is legal }] \\
&= \Pr_{A,t}[\ S(A,t) \neq error \mid (A,t) \text{ is legal }] \cdot \Pr_{A,t}[\ (A,t) \text{ is legal }] \geq \frac{1}{2\log^{c_s} N}.
\end{aligned}$$

In addition,

$$\begin{aligned}
&\Pr_{A,t}[S(A,t) \neq error] = E_A\left[\ \frac{|S(A)|}{N}\ \right] \\
&\leq \Pr_A\left[\ |S(A)| \geq \frac{N}{4\log^{c_s} N}\ \right] + \Pr_A\left[\ |S(A)| < \frac{N}{4\log^{c_s} N}\ \right] \cdot \frac{1}{4\log^{c_s} N} \\
&\leq \Pr_A\left[\ |S(A)| \geq \frac{N}{4\log^{c_s} N}\ \right] + \frac{1}{4\log^{c_s} N}.
\end{aligned}$$

[2] We could also consider randomized routines S, but this makes essentially no difference: there is always a way to fix the random coins of a randomized routine such that the resulting deterministic routine answers an equally large fraction of the inputs.

We complete the proof by combining the two inequalities. ∎

LEMMA 4.3. *Let $T \subseteq \{0,\ldots,N-1\}$ be a set such that $|T| > \frac{N}{s}$ for a certain s. Then, for any $q < \frac{N}{8s}$ there exists $q' \in \{q, 2q, \ldots, sq\}$ such that the number of pairs $t, t+q'$ that are both in T is $\Omega(\frac{N}{s^3})$.*

Proof. Define the partition of T into sets $T_0,\ldots,T_{q-1}$ as

$$T_k = \{i \mid i \in T, i \equiv k \pmod q\}.$$

At least $\frac{q}{2s}$ of the sets are of size at least $\frac{N}{2sq}$ since their union is T and $\frac{q}{2s}\cdot\frac{N}{q}+\frac{N}{2s} < |T|$. Let T_i be such a set and for $t \in T_i$ consider the values

$$t+q, t+2q, \ldots, t+4sq.$$

Therefore, the number of $t \in T_i$ such that none of these values is in T_i is less than $\frac{N}{4sq}$ because

$$|\{i \mid 0 \le i < N, i \equiv k \pmod q\}| = \frac{N}{q}.$$

Therefore, more than $|T_i| - \frac{N}{4sq} \ge \frac{N}{4sq}$ of the elements $t \in T_i$ are such that one of

$$t+q, t+2q, \ldots, t+4sq$$

is also in T_i. Summing over all sets T_i such that $|T_i| \ge \frac{N}{2sq}$, there are at least $\frac{N}{4sq}\cdot\frac{q}{2s} = \frac{N}{8s^2}$ elements $t \in T$ for which one of $t+q, t+2q, \ldots, t+4sq$ is also in T. Thus, there exists a $q' \in \{q, 2q, \ldots, 4sq\}$ such that the number of $t \in T$ for which $t+q' \in T$ is at least $\frac{N}{32s^3}$. ∎

DEFINITION 4.4. *A partial function $f : \{0,\ldots,N-1\} \to \{0,\ldots,N-1\}$ is called a matching if for all i such that $f(i)$ is defined, $f(i) \ne i$ and $f(f(i)) = i$. A matching is a q-matching if for all i such that $f(i)$ is defined, $|f(i)-i| = q$. We define an equal partition of the domain of a matching f by $A_1(f) = \{i \mid f(i)$ defined $\wedge\ f(i) > i\}$ and $A_2(f) = \{i \mid f(i)$ defined $\wedge\ f(i) < i\}$. The intersection of a matching f and a set $T \subseteq \{0,\ldots,N-1\}$ is the set $\{i \mid i \in T \wedge f(i) \in T\}$.*

For any q we define the following q-matchings:

$$f_q^1(t) = \begin{cases} t+q & t \bmod 2q < q,\ t+q < N, \\ t-q & t \bmod 2q \ge q,\ t-q \ge 0, \\ \textit{undefined} & \text{otherwise.} \end{cases}$$

$$f_q^2(t) = \begin{cases} t-q & t \bmod 2q < q,\ t-q \ge 0, \\ t+q & t \bmod 2q \ge q,\ t+q < N, \\ \textit{undefined} & \text{otherwise.} \end{cases}$$

LEMMA 4.5. *There exists a constant c_{m} such that for any integer $q < \frac{N}{\log^{c_{\mathsf{m}}} N}$ there exists a matching f among the $2\log^{c_{\mathsf{m}}} N$ matchings*

$$f_q^1, f_{2q}^1, \ldots, f_{\log^{c_{\mathsf{m}}} Nq}^1, f_q^2, f_{2q}^2, \ldots, f_{\log^{c_{\mathsf{m}}} Nq}^2$$

such that with probability at least $\frac{1}{\log^{c_{\mathsf{m}}} N}$ on the choice of A, the intersection of f and $S(A)$ is $\frac{N}{\log^{c_{\mathsf{m}}} N}$. We call such an f a good *matching.*

Proof. According to Lemma 4.2, $\frac{1}{4\log^{c_{\mathsf{s}}} N}$ of the possible values of A satisfy $|S(A)| > \frac{N}{4\log^{c_{\mathsf{s}}} N}$. For such A, Lemma 4.3 with $s = 4\log^{c_{\mathsf{s}}} N$ implies that there exists a value

$$q' \in \{q, 2q, \ldots, 4\log^{c_{\mathsf{s}}} N \cdot q\}$$

such that the number of pairs $t, t+q'$ that are both in $S(A)$ is $\Omega(\frac{N}{\log^{3c_s} N})$. Therefore, for such A and q', the size of the intersection of one of the matchings $f^1_{q'}, f^2_{q'}$ and $S(A)$ is $\Omega(\frac{N}{\log^{3c_s} N})$. This implies that one of the $8\log^{c_s} N$ matchings considered must have an intersection of size $\Omega(\frac{N}{\log^{3c_s} N})$ with at least $\frac{1}{32\log^{2c_s} N}$ of the possible values of A. We conclude the proof by choosing $c_m > 3c_s$. □

4.2. The quantum algorithm. We begin with the following simple claim.

CLAIM 4.6. *For any two basis states $|a\rangle$ and $|b\rangle$, $a \neq b$, there exists a routine such that, given the state $|a\rangle + e(\phi)|b\rangle$, outputs the state $|0\rangle + e(\phi)|1\rangle$.*

Proof. Consider the function f defined as $f(a) = 0, f(0) = a, f(b) = 1, f(1) = b$, and $f(i) = i$ otherwise. It is reversible and can therefore be implemented as a quantum routine. □

We now describe the main routine in the DCP algorithm.

LEMMA 4.7. *There exist routines R_1, R_2 such that given a q-matching f and an input for the DCP with failure parameter 1, either they output a bit or they fail. Conditioned on nonfailure, the probability of the bit being 1 is $\frac{1}{2} - \frac{1}{2}\cos(2\pi q\frac{d}{N})$ for R_1 and $\frac{1}{2} + \frac{1}{2}\sin(2\pi q\frac{d}{N})$ for R_2. Moreover, if f is a good matching, the success probability is $\Omega(\frac{1}{\log^{c_m} N})$.*

Proof. The routines begin by performing a Fourier transform on the last $\log N$ qubits of each input register. Consider one register. Assuming it is a good register, the resulting state is

$$\frac{1}{\sqrt{2N}}\sum_{i=0}^{N-1} e(ix/N)|0,i\rangle + \frac{1}{\sqrt{2N}}\sum_{i=0}^{N-1} e(i(x+d)/N)|1,i\rangle$$
$$= \frac{1}{\sqrt{2N}}\sum_{i=0}^{N-1} e(ix/N)(|0\rangle + e(id/N)|1\rangle)|i\rangle.$$

We measure the last $\log N$ qubits and let $a \in \{0,\ldots,N-1\}$ be the result. The state collapses to

$$\frac{1}{\sqrt{2}} e(ax/N)(|0\rangle + e(ad/N)|1\rangle)|a\rangle.$$

If it is a bad register, it is in the state $|b,x\rangle$, where both b and x are arbitrary. After the Fourier transform the state is

$$\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1} e(ix/N)|b,i\rangle,$$

and after measuring a in the last $\log N$ qubits, the state is $e(ax/N)|b,a\rangle$. Notice that in both cases any value a in $\{0,\ldots,N-1\}$ has an equal probability of being measured.

We choose the number of input registers to be r. Let $A = (a_1,\ldots,a_r)$ be the sequence of values measured in the above process. Notice that this sequence is uniform and hence can be used as an input to the average-case subset sum algorithm. In the following, we assume that s of the r registers are bad. Later we will claim that with good probability, none of the registers is bad. Yet, we have to show that even if one of the registers is bad, the routine does not return erroneous results. Without loss of generality, assume that the first s registers are bad. The resulting state is

$$\bigotimes_{i=1}^{s}[e(a_i x_i/N)|b_i,a_i\rangle] \bigotimes_{i=s+1}^{r}\left[\frac{1}{\sqrt{2}} e(a_i x_i/N)(|0\rangle + e(a_i d/N)|1\rangle)|a_i\rangle\right],$$

or, by omitting the multiplication by the fixed phase and the $r \cdot \lceil \log N \rceil$ fixed qubits,

$$\bigotimes_{i=1}^{s}[|b_i\rangle] \bigotimes_{i=s+1}^{r}\left[\frac{1}{\sqrt{2}}(|0\rangle + e(a_i d/N)|1\rangle)\right].$$

Denote these r qubits by $\bar{\alpha} = (\alpha_1, \ldots, \alpha_r)$.

We add $r+1$ new qubits, $\bar{\beta} = (\beta_1, \ldots, \beta_r)$ and γ. Let $t_{\bar{\alpha}}$ denote the sum $\sum_{i=1}^{r} \alpha_i a_i$. Next, we perform the following operations:

if $S(A, t_{\bar{\alpha}}) \neq \bar{\alpha} \ \vee \ S(A, f(t_{\bar{\alpha}})) = error$
  then *exit*
if $t_{\bar{\alpha}} \in A_1(f)$
  then $\begin{cases} \bar{\beta} \leftarrow \bar{\alpha} \\ \gamma \leftarrow 1 \end{cases}$
  else if $t_{\bar{\alpha}} \in A_2(f)$
  then $\begin{cases} \bar{\beta} \leftarrow S(A, f(t_{\bar{\alpha}})) \\ \gamma \leftarrow 1 \end{cases}$
  else *exit*

In order to describe the state after the above procedure, we define the following subsets of $\{0,1\}^r$:

$$\begin{aligned} M &= \{\bar{\alpha} \in \{0,1\}^r \mid \alpha_1 = b_1, \ldots, \alpha_s = b_s\}, \\ L &= \{\bar{\alpha} \in M \mid t_{\bar{\alpha}} \in A_1(f) \ \wedge \ S(A, t_{\bar{\alpha}}) = \bar{\alpha} \ \wedge S(A, f(t_{\bar{\alpha}})) \neq error\}, \\ R &= \{\bar{\alpha} \in M \mid t_{\bar{\alpha}} \in A_2(f) \ \wedge \ S(A, t_{\bar{\alpha}}) = \bar{\alpha} \ \wedge S(A, f(t_{\bar{\alpha}})) \neq error\}. \end{aligned}$$

Using the order $|\bar{\alpha}, \bar{\beta}, \gamma\rangle$, the resulting state is

$$\begin{aligned} &\frac{1}{\sqrt{2^{r-s}}}\left(\sum_{\bar{\alpha}\in M-L-R} e\left(\langle\bar{\alpha},\bar{a}\rangle\frac{d}{N}\right)|\bar{\alpha},\bar{0},0\rangle\right. \\ &\qquad\left. + \sum_{\bar{\alpha}\in L} e\left(\langle\bar{\alpha},\bar{a}\rangle\frac{d}{N}\right)|\bar{\alpha},\bar{\alpha},1\rangle + \sum_{\bar{\alpha}\in R} e\left(\langle\bar{\alpha},\bar{a}\rangle\frac{d}{N}\right)|\bar{\alpha},S(A,f(t_{\bar{\alpha}})),1\rangle\right) \\ &= \frac{1}{\sqrt{2^{r-s}}}\left(\sum_{\bar{\alpha}\in M-L-R} e\left(\langle\bar{\alpha},\bar{a}\rangle\frac{d}{N}\right)|\bar{\alpha},\bar{0},0\rangle\right. \\ &\qquad\left. + \sum_{\bar{\alpha}\in L}\left(e\left(\langle\bar{\alpha},\bar{a}\rangle\frac{d}{N}\right)|\bar{\alpha},\bar{\alpha},1\rangle + e\left(\langle S(A,f(t_{\bar{\alpha}})),\bar{a}\rangle\frac{d}{N}\right)|S(A,f(t_{\bar{\alpha}})),\bar{\alpha},1\rangle\right)\right) \\ &= \frac{1}{\sqrt{2^{r-s}}}\left(\sum_{\bar{\alpha}\in M-L-R} e\left(\langle\bar{\alpha},\bar{a}\rangle\frac{d}{N}\right)|\bar{\alpha},\bar{0},0\rangle\right. \\ &\qquad\left. + \sum_{\bar{\alpha}\in L} e\left(\langle\bar{\alpha},\bar{a}\rangle\frac{d}{N}\right)(|\bar{\alpha}\rangle + e(q\cdot\frac{d}{N})|S(A,f(t_{\bar{\alpha}}))\rangle)|\bar{\alpha},1\rangle\right). \end{aligned}$$

Now we measure $\bar{\beta}$ and γ. If $\gamma = 0$, the routine failed. Otherwise, the state of $\bar{\alpha}$ is (omitting the fixed $\bar{\beta}$ and γ)

$$\frac{1}{\sqrt{2}}\left(|\bar{\beta}\rangle + e\left(q\cdot\frac{d}{N}\right)|S(A,f(t_{\bar{\beta}}))\rangle\right).$$

Notice that since $\bar{\beta}$ is known and $S(A, f(t_{\bar{\beta}}))$ can be easily found by calling S, we can transform this state to the state

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e\left(q \cdot \frac{d}{N}\right)|1\rangle\right)$$

by using Claim 4.6. By omitting some qubits, we can assume that this is a state on one qubit. By using the Hadamard transform the state becomes

$$\frac{1}{2}\left(\left(1 + e\left(q \cdot \frac{d}{N}\right)\right)|0\rangle + \left(1 - e\left(q \cdot \frac{d}{N}\right)\right)|1\rangle\right).$$

We measure the qubit, and the probability of measuring 1 is

$$\frac{1}{4}\left|1 - e\left(q \cdot \frac{d}{N}\right)\right|^2 = \frac{1}{4}\left(2 - 2\cos\left(2\pi q \cdot \frac{d}{N}\right)\right) = \frac{1}{2} - \frac{1}{2}\cos\left(2\pi q \cdot \frac{d}{N}\right).$$

This completes the description of R_1. The routine R_2 applies the transform

$$\begin{pmatrix} 1 & 0 \\ 0 & \imath \end{pmatrix}$$

before the Hadamard transform and thus the state becomes

$$\frac{1}{2}\left(\left(1 + e\left(1/4 + q \cdot \frac{d}{N}\right)\right)|0\rangle + \left(1 - e\left(1/4 + q \cdot \frac{d}{N}\right)\right)|1\rangle\right),$$

and the probability of measuring 1 becomes

$$\frac{1}{2} - \frac{1}{2}\cos\left(\pi/2 + 2\pi q \cdot \frac{d}{N}\right) = \frac{1}{2} + \frac{1}{2}\sin\left(2\pi q \cdot \frac{d}{N}\right).$$

From the previous description, it is clear that the probability of measuring 1 conditioned on a nonfailure is correct. Thus, it remains to prove that when f is a good matching the failure probability is low. The success probability equals the probability of measuring $\gamma = 1$, which is $|L \cup R|/2^{r-s}$. Assume that none of the r registers is bad. Then $|L \cup R|/2^{r-s} = |L \cup R|/2^r$ and $L \cup R$ becomes

$$\{\bar{\alpha} \in \{0,1\}^r \mid t_{\bar{\alpha}} \in A_1(f) \cup A_2(f) \ \wedge \ S(A, t_{\bar{\alpha}}) = \bar{\alpha} \ \wedge \ S(A, f(t_{\bar{\alpha}})) \neq error\}.$$

Notice that the size of this set equals

$$|\{t \mid t \in S(A) \ \wedge \ f(t) \in S(A)\}|,$$

which, according to the definition of a good matching, is at least $\frac{N}{\log^{c_m} N}$. Therefore the probability of success conditioned on all of the registers being good is

$$|L \cup R|/2^r = \frac{1}{2^{c_r}\log^{c_m} N} = \Omega\left(\frac{1}{\log^{c_m} N}\right).$$

This concludes the proof since with probability at least

$$\left(1 - \frac{1}{\log N}\right)^r = \left(1 - \frac{1}{\log N}\right)^{\log N + c_r} = \Omega(1)$$

none of the registers is bad. ∎

CLAIM 4.8. *Given an approximation x of $\sin\phi$ and an approximation y of $\cos\phi$ with additive error ϵ, we can find $\phi \bmod 2\pi$ up to an additive error of $O(\epsilon)$.*

Proof. Assume $y \geq 0$ and let $z = \frac{x}{1+y}$. A simple calculation shows that z is an estimate of $\frac{\sin\phi}{1+\cos\phi}$ up to an additive error of at most 4ϵ. The estimate on ϕ is $2\arctan z$. Since the absolute value of the differential of arctan is at most 1, this is an estimate of $2\arctan(\frac{\sin\phi}{1+\cos\phi}) = \phi$ with an additive error of at most 8ϵ. When $y < 0$ we compute an estimate of $2\operatorname{arccot}(\frac{\sin\phi}{1-\cos\phi}) = \phi$. ∎

LEMMA 4.9. *There exists a routine R_3 such that with probability exponentially close to 1, given any $q < \frac{N}{\log^{c_{\mathsf{m}}} N}$, finds a value $q' \in \{q, \ldots, \log^{c_{\mathsf{m}}} N \cdot q\}$ and an estimate x such that*

$$x \in \left[q'd - \frac{N}{\log^{c_{\mathsf{m}}+1} N}, q'd + \frac{N}{\log^{c_{\mathsf{m}}+1} N}\right] \pmod N.$$

Proof. Assume we are given a q'-matching f. We call routines R_1 and R_2 $\log^{3c_{\mathsf{m}}+4} N$ times. If the number of successful calls to one of the routines is less than $\log^{2c_{\mathsf{m}}+3} N$, we fail. Otherwise, let $x \in [0,1]$ be the average of the successful calls to R_1 and let $y \in [0,1]$ be the average of the successful calls to R_2. According to the Chernoff bound,

$$\Pr\left[\left|x - \left(\frac{1}{2} - \frac{1}{2}\cos\left(2\pi q' \cdot \frac{d}{N}\right)\right)\right| > \frac{1}{c_{\mathsf{e}} \log^{c_{\mathsf{m}}+1} N}\right] < 2e^{-2\log^{2c_{\mathsf{m}}+3} N/(c_{\mathsf{e}}{}^2 \log^{2c_{\mathsf{m}}+2} N)},$$

which is exponentially low in $\log N$ for any constant $c_{\mathsf{e}} > 0$. A similar bound holds for y. Hence, we can assume that $x' = 1 - 2x$ and $y' = 2y - 1$ are approximations of $\cos(2\pi q' \cdot \frac{d}{N})$ and of $\sin(2\pi q' \cdot \frac{d}{N})$, respectively, up to an additive error of $\frac{2}{c_{\mathsf{e}} \log^{c_{\mathsf{m}}+1} N}$. According to Claim 4.8, this translates to an estimate of $q' \cdot \frac{d}{N} \bmod 1$ with an additive error of $\frac{1}{\log^{c_{\mathsf{m}}+1} N}$ for c_{e} large enough.

By repeating the above procedure with all the matchings that appear in Lemma 4.5, we are guaranteed to find a good matching. According to Lemma 4.7, a call to routine R_1 or to routine R_2 with a good matching succeeds with probability at least $c_{\mathsf{g}} \frac{1}{\log^{c_{\mathsf{m}}} N}$ for a certain $c_{\mathsf{g}} > 0$. The probability that none of $\log^{c_{\mathsf{m}}+1} N$ calls to the subroutine succeeds is

$$\left(1 - c_{\mathsf{g}} \frac{1}{\log^{c_{\mathsf{m}}} N}\right)^{\log^{c_{\mathsf{m}}+1} N},$$

which is exponentially small. Thus, for one of the matchings, with probability exponentially close to 1, we have $\log^{2c_{\mathsf{m}}+3} N$ successful calls to routines R_1 and R_2 and routine R_3 is successful. ∎

We conclude the proof of Theorem 1.3 with a description of the algorithm for finding d. We begin by using routine R_3 with the value 1 to obtain an estimate x_1 and a value $\hat{q} \leq \log^{c_{\mathsf{m}}} N$ such that

$$x_1 \in \left[d' - \frac{N}{\log^{c_{\mathsf{m}}+1} N}, d' + \frac{N}{\log^{c_{\mathsf{m}}+1} N}\right] \pmod N,$$

where d' denotes $(d\hat{q} \bmod N)$. In the following we find d' exactly by calling R_3 with multiples of $\hat{q}$. The algorithm works in stages. In stage i we have an estimate x_i and

a value q_i. The invariant we maintain is

$$x_i \in \left[q_i d' - \frac{N}{\log^{c_m+1} N},\ q_i d' + \frac{N}{\log^{c_m+1} N} \right] \pmod{q_i N}.$$

We begin with x_1 as above and $q_1 = 1$. Assume that the invariant holds in stage i. We use routine R_3 with the value $2q_i\hat{q}$ to obtain an estimate x with a value

$$q' \in \{2q_i\hat{q}, 4q_i\hat{q}, \ldots, 2\log^{c_m} N \cdot q_i\hat{q}\}$$

such that

$$x \in \left[q_{i+1} d' - \frac{N}{\log^{c_m+1} N},\ q_{i+1} d' + \frac{N}{\log^{c_m+1} N} \right] \pmod{N},$$

where $q_{i+1} = q'/\hat{q}$. Notice that our previous estimate x_i satisfies

$$\frac{q_{i+1}}{q_i} x_i \in \left[q_{i+1} d' - \frac{2N}{\log N}, q_{i+1} d' + \frac{2N}{\log N} \right] \pmod{q_{i+1} N}.$$

Since this range is much smaller than N, we can combine the estimate x on $(q_{i+1}d' \bmod N)$ and the estimate $\frac{q_{i+1}}{q_i} x_i$ on $(q_{i+1}d' \bmod q_{i+1}N)$ to obtain x_{i+1} such that

$$x_{i+1} \in \left[q_{i+1} d' - \frac{N}{\log^{c_m+1} N},\ q_{i+1} d' + \frac{N}{\log^{c_m+1} N} \right] \pmod{q_{i+1} N}.$$

The last stage is when $q_i \geq \frac{4N}{\log^{c_m+1} N}$. Then d' can be found by rounding $\frac{x_i}{q_i}$ to the nearest integer. Given d' there are at most $\hat{q} \leq \log^{c_m} N$ possible values for q. Since this is only a polynomial number of options, we can output one randomly.

Acknowledgments. I would like to thank Dorit Aharonov, Noga Alon, Andris Ambainis, Irit Dinur, Sean Hallgren, Alexei Kitaev, Hartmut Klauck, Greg Kuperberg, Ashwin Nayak, Cliff Smyth, and Avi Wigderson for many helpful discussions and comments. I also thank the anonymous referees for their comments.

REFERENCES

[1] M. Ajtai, *Generating hard instances of lattice problems*, in Proceedings of the 28th ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 99–108. Available online from ECCC at http://www.uni-trier.de/eccc/.

[2] M. Ajtai and C. Dwork, *A public-key cryptosystem with worst-case/average-case equivalence*, in Proceedings of the 29th ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 284–293. Available online from ECCC at http://www.uni-trier.de/eccc/.

[3] J.-Y. Cai, *A New Transference Theorem and Applications to Ajtai's Connection Factor*, Electronic Colloquium on Computational Complexity (ECCC), Report TR98-005, 1998. Available online from ECCC at http://www.eccc.uni-trier.de/eccc/.

[4] J.-Y. Cai and A. Nerurkar, *An improved worst-case to average-case connection for lattice problems*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 468–477.

[5] W. van Dam, S. Hallgren, and L. Ip, *Quantum algorithms for hidden shift problems*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2003, pp. 489–498.

[6] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proc. Roy. Soc. London Ser. A, 439 (1992), pp. 553–558.

[7] M. Ettinger and P. Høyer, *On quantum algorithms for noncommutative hidden subgroups*, Adv. in Appl. Math., 25 (2000), pp. 239–251.

[8] K. Friedl, G. Ivanyos, F. Magniez, M. Santha, and P. Sen, *Hidden translation and orbit coset in quantum computing*, in Proceedings of the 35th ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 1–9.

[9] M. Grigni, L. J. Schulman, M. Vazirani, and U. V. Vazirani, *Quantum mechanical algorithms for the nonabelian hidden subgroup problem*, in Proceedings of the 33rd ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 68–74.

[10] L. Grover and T. Rudolph, *Creating Superpositions That Correspond to Efficiently Integrable Probability Distributions*, preprint, 2003. Available online from http://arxiv.org/abs/quant-ph/0208112.

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

[12] S. Hallgren, *Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem*, in Proceedings of the 34th ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 653–658.

[13] S. Hallgren, A. Russell, and A. Ta-Shma, *Normal subgroup reconstruction and quantum computation using group representations*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 627–635.

[14] R. Impagliazzo and M. Naor, *Efficient cryptographic schemes provably as secure as subset sum*, J. Cryptology, 9 (1996), pp. 199–216.

[15] K. Johannes, S. Uwe, and T. Jacobo, *The Graph Isomorphism Problem: Its Structural Complexity*, Birkhäuser Boston, Boston, 1993.

[16] R. Kannan, L. Lovász, and M. Simonovits, *Random walks and an $O^*(n^5)$ volume algorithm for convex bodies*, Random Structures Algorithms, 11 (1997), pp. 1–50.

[17] A. Y. Kitaev, *Quantum Measurements and the Abelian Stabilizer Problem*, preprint, 1995. Available online from http://arxiv.org/abs/quant-ph/9511026.

[18] G. Kuperberg, *A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem*, preprint, 2003. Available online from http://arxiv.org/abs/quant-ph/0302112.

[19] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 515–534.

[20] D. Micciancio, *Improved cryptographic hash functions with worst-case/average-case connection*, in Proceedings of the 34th ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 609–618.

[21] D. Micciancio and S. Goldwasser, *Complexity of Lattice Problems: A Cryptographic Perspective*, Kluwer Int. Ser. Engrg. Comput. Sci. 671, Kluwer Academic, Boston, 2002.

[22] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.

[23] O. Regev, *New lattice based cryptographic constructions*, in Proceedings of the 35th ACM Symposium on Theory of Computing, San Diego, CA, ACM, New York, 2003, pp. 407–416.

[24] M. Rötteler and T. Beth, *Polynomial-Time Solution to the Hidden Subgroup Problem for a Class of Non-Abelian Groups*, preprint, 1998. Available online from http://arxiv.org/abs/quant-ph/9812070.

[25] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.

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

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 761–782

COMPACTION, RETRACTION, AND CONSTRAINT SATISFACTION*

NARAYAN VIKAS†

Abstract. In this paper, we show a very close relationship among the compaction, retraction, and constraint satisfaction problems in the context of reflexive and bipartite graphs. The compaction and retraction problems are special graph coloring problems, and the constraint satisfaction problem is well known to have an important role in artificial intelligence. The relationships we present provide evidence that, similar to the retraction problem, it is likely to be difficult to determine whether for every fixed reflexive or bipartite graph, the compaction problem is polynomial time solvable or NP-complete. In particular, the relationships that we present relate to a long-standing open problem concerning the equivalence of the compaction and retraction problems.

Key words. computational complexity, graph, coloring, homomorphism, retraction, compaction, constraint satisfaction

AMS subject classifications. 68Q25, 68R10

DOI. 10.1137/S0097539701397801

1. Introduction.

We first introduce the following definitions and problems, and then describe the motivation and results.

1.1. Definitions.

The pair of vertices forming an edge in a graph are called the *endpoints* of the edge. An edge is said to be *incident* with a vertex v in a graph if v is an endpoint of the edge. An edge of a graph with the same endpoints is called a *loop*. We say that a vertex v of a graph has a loop if vv is an edge of the graph. A *reflexive graph* is a graph in which every vertex has a loop. An *irreflexive graph* is a graph in which no vertex has a loop. Any graph, in general, is a *partially reflexive graph*, meaning that its individual vertices may or may not have loops. Thus reflexive and irreflexive graphs are special partially reflexive graphs. A *bipartite graph* G is a graph whose vertex set can be partitioned into two distinct subsets G_A and G_B, such that each edge of G has one endpoint in G_A and the other endpoint in G_B; we say that (G_A, G_B) is a bipartition of G. Thus a bipartite graph is irreflexive by definition. When we do not mention the terms reflexive, irreflexive, or bipartite, the corresponding graph may be assumed to be a partially reflexive graph. We denote the vertex set and the edge set of a graph G by $V(G)$ and $E(G)$ respectively. A vertex u is said to be *adjacent* to a vertex v in a graph if uv is an edge of the graph; if u is adjacent to v then v is also adjacent to u. If a vertex u is adjacent to a vertex v then u is said to be a *neighbor* of v, and v is said to be a neighbor of u. A graph in which each pair of distinct vertices are adjacent is called a *complete graph*. We denote an irreflexive complete graph with k vertices by K_k. A bipartite graph, with bipartition (G_A, G_B), in which every vertex in G_A is adjacent to every vertex in G_B is called a *complete bipartite graph*. A *path* of length $k-1$ is a graph containing k distinct vertices, say $v_0, v_1, v_2, \ldots, v_{k-1}$, such that $v_0v_1, v_1v_2, \ldots, v_{k-2}v_{k-1}$ are all the nonloop edges of the graph, $k \geq 1$; we may write such a path as $v_0v_1v_2 \ldots v_{k-1}$. A *cycle* of length k, called a *k-cycle*, is a graph containing k distinct vertices, say $v_0, v_1, v_2, \ldots, v_{k-1}$, such that

*Received by the editors November 12, 2001; accepted for publication October 22, 2002; published electronically May 5, 2004.
http://www.siam.org/journals/sicomp/33-4/39780.html
†School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6 (vikas@cs.sfu.ca).

$v_0v_1, v_1v_2, \ldots, v_{k-2}v_{k-1}, v_{k-1}v_0$ are all the nonloop edges of the graph, $k \geq 3$; we may write such a cycle as $v_0v_1v_2 \ldots v_{k-1}v_0$.

Let G be a graph. A vertex v of G is said to be an *isolated vertex* of G if v is not adjacent to any other vertex v' of G, $v \neq v'$ (note that an isolated vertex may have a loop). When a set S is an argument of a mapping f, we define $f(S) = \{f(s)|s \in S\}$. We use $|S|$ to denote the cardinality of a set S. The *distance* between a pair of vertices u and v in G, denoted by $d_G(u, v)$ or $d_G(v, u)$, is the length of a shortest path from u to v in G if u and v are connected in G; we define $d_G(u, v)$ (and $d_G(v, u)$) to be infinite if u and v are disconnected in G. The *diameter* of G is the maximum distance between any two vertices in G, i.e., $\max \{d_G(u, v)|u, v \in V(G)\}$, where $\max A$ gives the maximum element in a set A. The distance between two sets X and Y of vertices in G, denoted by $d_G(X, Y)$ or $d_G(Y, X)$, is the minimum distance between any vertex of X and any vertex of Y in G, i.e., $d_G(X, Y) = \min \{d_G(x, y)|x \in X, y \in Y\}$, where $\min A$ gives the minimum element in a set A. If a set has only one vertex, we may just write the vertex instead of the set. A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If H is a subgraph of G such that H contains all the edges of G that have both endpoints in $V(H)$ then H is called the subgraph of G *induced by* $V(H)$, and we say that H is an *induced subgraph* of G. Given an induced subgraph H of G, we denote by $G - H$, the subgraph obtained by deleting from G the vertices of H together with the edges incident with them; thus $G - H$ is a subgraph of G induced by $V(G) - V(H)$. A *chordal graph* is a graph which does not contain any induced cycle of length greater than three. A *chordal bipartite graph* is a bipartite graph which does not contain any induced cycle of length greater than four. In the following, let G and H be graphs.

A *homomorphism* $f : G \to H$, of G to H, is a mapping f of the vertices of G to the vertices of H, such that $f(g)$ and $f(g')$ are adjacent vertices of H whenever g and g' are adjacent vertices of G. If there exists a homomorphism of G to H then G is said to be *homomorphic* to H. Note that for any homomorphism $f : G \to H$, if a vertex v of G has a loop then the vertex $f(v)$ of H necessarily also has a loop. If G is irreflexive then clearly G is k-colorable if and only if G is homomorphic to K_k. Thus the concept of a homomorphism generalizes the concept of a k-colorability.

A *compaction* $c : G \to H$, of G to H, is a homomorphism of G to H, such that for every vertex x of H, there exists a vertex v of G with $c(v) = x$, and for every edge hh' of H, $h \neq h'$, there exists an edge gg' of G with $c(g) = h$ and $c(g') = h'$. Notice that the first part of the definition for compaction (the requirement for every vertex x of H) is relevant only if H has isolated vertices. If there exists a compaction of G to H then G is said to *compact* to H. Given a compaction $c : G \to H$, if for a vertex v of G, we have $c(v) = x$, where x is a vertex of H, then we say that the vertex v of G *covers the vertex* x of H under c; and if for an edge gg' of G, we have $c(\{g, g'\}) = \{h, h'\}$, where hh' is an edge of H, then we say that the edge gg' of G *covers the edge* hh' of H under c (note that in the definition of compaction, it is not necessary that a loop of H be covered by any edge of G under c).

We note that the notion of a *homomorphic image* used in [Harary, 1969] (also cf. [Hell and Miller, 1979]) coincides with the notion of a compaction in the case of irreflexive graphs (i.e., when G and H are irreflexive in the above definition for compaction).

A *retraction* $r : G \to H$, of G to H, with H as an induced subgraph of G, is a homomorphism of G to H, such that $r(h) = h$, for every vertex h of H. If there exists a retraction of G to H then G is said to *retract* to H. Note that every retraction $r : G \to H$ is necessarily also a compaction but not vice versa.

For each vertex v of G, let $L(v)$ be a list of vertices of H. We denote by L the entire set of lists $L(v)$ for the vertices v of G. A *list homomorphism* $l : G \to H$, of G to H, with respect to L is a homomorphism of G to H, such that $l(v) \in L(v)$, for every vertex v of G. If $l : G \to H$ is a list homomorphism with respect to L then we say that $l : G \to H$ is a list-L-homomorphism.

Let L be the set of lists as defined above. The *consistency test* for G with respect to H and L produces a set L^* of lists $L^*(v) \subseteq L(v)$, for all $v \in V(G)$, such that for every edge ab of G, if a vertex $h \in L^*(a)$ then there exists a vertex $h' \in L^*(b)$ such that hh' is an edge of H, and if a vertex $z \in L^*(b)$ then there exists a vertex $z' \in L^*(a)$ such that zz' is an edge of H; the set L^* is obtained in such a way that minimum number of vertices are removed from $L(v)$, for all $v \in V(G)$. Thus for an edge ab of G, if a vertex h in $L(a)$ has no neighbor h' in $L(b)$, with $hh' \in E(H)$, then the consistency test will delete h from $L(a)$, and similarly if a vertex z in $L(b)$ has no neighbor z' in $L(a)$, with $zz' \in E(H)$, then the consistency test will delete z from $L(b)$. Due to such deletions, the consistency test can propagate further such deletions from other lists with respect to other edges of G even if those edges have been considered before. The process of the consistency test stops after no further such deletions are possible, at which point we obtain L^*, which is the final modified set L of lists after all the deletions. Note that there exists a list-L-homomorphism of G to H if and only if there exists a list-L^*-homomorphism of G to H. The consistency test we have described is in essence the *arc consistency test* used in artificial intelligence [Mackworth, 1977]. It follows from the result of [Mackworth and Freuder, 1985] that if H is fixed then L^* can be obtained in time linear in the size of G. We say that the consistency test for G with respect to H and L *succeeds* if $L^*(v) \neq \phi$, for all $v \in V(G)$; otherwise we say that it does not succeed. We shall be using the consistency test in our proofs. The consistency test has been an important tool in artificial intelligence where it has been widely used for a long time, including in earlier studies on constraint satisfaction problems [Montanari, 1974]. The consistency test has been increasingly used in graph homomorphism problems over the last years including [Gutjahr, Welzl, and Woeginger, 1992], [Hell, Nesetril, and Zhu, 1996], [Vikas, 2002].

An *identification* of two distinct vertices u and v of G is an execution of the following steps (1), (2), and (3), resulting in a new graph: (1) For every nonloop edge uu' of G, if vu' is not an edge of G then we add the edge vu' to G (note that if uv is an edge of G then $u' = v$ and we will have the loop vv). (2) If u has a loop then v is also made to have a loop if it does not already have one. (3) Delete the vertex u together with the edges incident with u from G.

An *isomorphism* $f : G \to H$, of G to H, is a mapping f of the vertices of G to the vertices of H, such that f is one-to-one and onto, and $f(g)$ and $f(g')$ are adjacent vertices of H if and only if g and g' are adjacent vertices of G (we are not assuming more than one edge between endpoints in any graph). If there exists an isomorphism of G to H then G is said to be *isomorphic* to H. An *automorphism* of G is an isomorphism of G to itself.

We may call a polynomial transformation under Turing reduction, just a polynomial transformation, and a polynomial equivalence under Turing reduction, just a polynomial equivalence, but the context will be clear in the proofs where we explicitly mention Turing reduction if Turing reduction is used.

1.2. Homomorphism, compaction, retraction, and constraint satisfaction problems. The problem of deciding the existence of a homomorphism to a fixed graph H, called the *homomorphism problem for* H, also known as the *H-coloring*

problem, and denoted as *H-COL*, asks whether or not an input graph G is homomorphic to H. If a vertex h of a graph H has a loop then every graph G is trivially homomorphic to H, as we will have a homomorphism $f : G \to H$, with $f(v) = h$, for all $v \in V(G)$. Thus the problem *H-COL* is interesting only if H is irreflexive. Further, among irreflexive graphs H, if H is bipartite then the problem *H-COL* is again trivial, as explained below. Note that only a bipartite graph may be homomorphic to another bipartite graph. If H is a bipartite graph containing an edge, say hh', then clearly every bipartite graph G is homomorphic to H, as we will have a homomorphism $f : G \to H$, with $f(a) = h$ and $f(b) = h'$, for all $a \in G_A$, $b \in G_B$, where (G_A, G_B) is a bipartition of G. If H is a graph containing no edge then clearly only graphs with no edge will be homomorphic to H. Thus a graph G is homomorphic to a bipartite graph H if and only if G is also bipartite, and H has an edge if G has an edge. Thus the problem *H-COL* is interesting only if H is an irreflexive nonbipartite graph. It has been shown in [Hell and Nesetril, 1990] that *H-COL* is NP-complete for any fixed irreflexive nonbipartite graph H, thus providing a complete complexity classification of *H-COL*. Note that the classic k-colorability problem is a special case of the problem *H-COL* when H is K_k and the input graph G is irreflexive.

The problem of deciding the existence of a compaction to a fixed graph H, called the *compaction problem for H*, and denoted as *COMP-H*, asks whether or not an input graph G compacts to H.

When both G and H are input graphs (i.e., H is not fixed), and H is reflexive, the problem of deciding whether or not G compacts to H has been studied in [Karabeg and Karabeg, 1991, 1993]. Note that unlike the problem *H-COL*, the problem *COMP-H* is still interesting if H has a loop or H is bipartite.

The problem of deciding the existence of a retraction to a fixed graph H, called the *retraction problem for H*, and denoted as *RET-H*, asks whether or not an input graph G, containing H as an induced subgraph, retracts to H. Note that H is a particular copy in G. It is possible that G contains another induced subgraph H' that is isomorphic to H but distinct from H, and it may be the case that G retracts to H but not to H', and vice versa.

Retraction problems have been of continuing interest in graph theory for a long time and have been studied in various literature including [Hell, 1972, 1974], [Nowakowski and Rival, 1979], [Pesch and Poguntke, 1985], [Bandelt, Dahlmann, and Schutte, 1987], [Hell and Rival, 1987], [Pesch, 1988], [Feder and Winkler, 1988], [Bandelt, Farber, and Hell, 1993], [Feder and Hell, 1998], [Feder, Hell, and Huang, 1999].

Note that the graph H for the problems *H-COL*, *COMP-H*, and *RET-H* is assumed to be fixed by default even if not explicitly mentioned.

We now define the constraint satisfaction problem after introducing the following terms. The *domain* of a variable is a set of permissible values that the variable can assume. A *relation* R on k variables $x_1, x_2, \ldots, x_k$, denoted by $R(x_1, x_2, \ldots, x_k)$, imposes constraints on the values that the variables $x_1, x_2, \ldots, x_k$ can assume from their domains; R is a *relation name* of *arity* k, where we assume that k is fixed, $R(x_1, x_2, \ldots, x_k)$ is a *tuple* of R, and R can be defined on several tuples. A tuple $R(a_1, a_2, \ldots, a_k)$ is said to be fixed if a_i is fixed, for all $i = 1, 2, \ldots, k$. A relation R is said to be a *fixed relation* if all of its tuples are fixed.

Let T be a set of fixed relations, and suppose that the cardinality of T is fixed. We call T a *fixed template*. Let S be a finite set of relations such that every relation name in S is also in T with the same arity. Let V_S be the set of all variables in any tuple of S. For each variable x in V_S, let $D(x)$ be the domain of x. The domains of

the variables in V_S will be assumed to be associated with the set S, and we shall not mention it explicitly. The set S is said to be *satisfied* with respect to the template T if there exists a mapping f from S to T such that for every tuple $R(x_1, x_2, \ldots, x_{k_R})$ of every relation R, if $R(x_1, x_2, \ldots, x_{k_R})$ is in S then $f(R(x_1, x_2, \ldots, x_{k_R}))$ defined to be $R(f(x_1), f(x_2), \ldots, f(x_{k_R}))$ is in T, where k_R denotes the arity of R, $f(x_i) \in D(x_i)$ if x_i is a variable, and $f(x_i) = x_i$ if x_i is fixed, for all $i = 1, 2, \ldots, k_R$. The *constraint satisfaction problem with respect to* T, denoted as *CSP-T*, takes S as an instance and asks the question whether or not S is satisfied with respect to T.

The constraint satisfaction problem is known to be of great interest in artificial intelligence. Many problems can be viewed as constraint satisfaction problems. We give an example here to demonstrate how the H-coloring problem can be viewed as the constraint satisfaction problem. Let H be a fixed graph with vertices $h_1, h_2, \ldots, h_p$. Let a graph G with vertices $v_1, v_2, \ldots, v_n$ be an instance of *H-COL*. Define T to be a fixed template containing a single relation E such that $E(h_i, h_j)$ is a tuple in T if and only if $h_i h_j$ is an edge of H, for all $i, j = 1, 2, \ldots, p$. Define S to contain the relation E such that $E(v_i, v_j)$ is a tuple in S if and only if $v_i v_j$ is an edge of G, for all $i, j = 1, 2, \ldots, n$. Define the domain of v_i to be $V(H)$, i.e., $\{h_1, h_2, \ldots, h_p\}$, for all $i = 1, 2, \ldots, n$. Thus S is an instance of *CSP-T*. Clearly, there exists a homomorphism of G to H if and only if S is satisfied with respect to T.

The template T for the problem *CSP-T* is assumed to be fixed by default even if not explicitly mentioned. Clearly, for a given template T, the problem *CSP-T* is in NP. As mentioned earlier, the problem *H-COL* is known to be NP-complete for all irreflexive nonbipartite graphs H [Hell and Nesetril, 1990]. We gave above a polynomial transformation from *H-COL* to *CSP-T*, implying that *CSP-T* is NP-complete for certain T. It is easy to see that there are also templates T for which *CSP-T* is polynomial time solvable. It is thought to be likely difficult to determine whether for every template T, the problem *CSP-T* is polynomial time solvable or NP-complete. Issues related to the constraint satisfaction problem have also been considered in [Feder and Vardi, 1993, 1998]. A very close relationship between the constraint satisfaction problem and the retraction problem for bipartite graphs has also been shown in [Feder and Vardi, 1998]. We shall describe this result later.

1.3. Motivation and results. It is not difficult to show that for every fixed graph H, if *RET-H* is solvable in polynomial time then *COMP-H* is also solvable in polynomial time. Is the converse true? This was also asked, in the context of reflexive graphs, by Peter Winkler in 1988 (personal communication, cf. [Feder and Winkler, 1988]). Thus the question is whether *RET-H* and *COMP-H* are polynomially equivalent for every fixed graph H. The answer to this is not known even when H is reflexive or bipartite. In this paper, we show that for every fixed reflexive (bipartite) graph H, there exists a fixed reflexive (bipartite) graph H' such that *RET-H* and *COMP-H'* are polynomially equivalent.

Using our above result and results of [Feder and Hell, 1998] and [Feder and Vardi, 1998], we establish that for every fixed template T of a set of relations, there exists a fixed reflexive (bipartite) graph H such that the constraint satisfaction problem *CSP-T* and the compaction problem *COMP-H* are polynomially equivalent. Since it is thought to be likely difficult to determine whether for every fixed template T, the problem *CSP-T* is polynomial time solvable or NP-complete, we thus provide evidence that it is likely to be difficult to determine whether for every fixed reflexive (bipartite) graph H, the problem *COMP-H* is polynomial time solvable or NP-complete. Similar evidence has been shown for *RET-H* in [Feder and Hell, 1998] in the case of fixed

reflexive graphs H, and in [Feder and Vardi, 1998] in the case of fixed bipartite graphs H.

We however have results giving a complete complexity classification of *COMP-H* and *RET-H* when H has four or fewer vertices, i.e., for every graph H with at most four vertices (including when H is partially reflexive), we determine whether *COMP-H* is polynomial time solvable or NP-complete, and whether *RET-H* is polynomial time solvable or NP-complete. The complexity classification of *COMP-H* and *RET-H* do not differ for such graphs H.

We have more results showing that for several graphs H, the problems *RET-H* and *COMP-H* are polynomially equivalent. We mention below a few classes of such graphs. We do not know of any graph H for which the complexity classification of *RET-H* and *COMP-H* differ.

It is known that *RET-H* is NP-complete when H is a reflexive k-cycle, for all $k \geq 4$, cf. [Feder and Hell, 1998], G. MacGillivray, 1988 (personal communication), and for $k = 4$, also [Feder and Winkler, 1988]. It is shown in [Vikas, 1999, 2003] that *COMP-H* is NP-complete when H is a reflexive k-cycle, for all $k \geq 4$. In particular, for $k = 4$, this result in [Vikas, 1999, 2003] solves a widely publicized open problem posed by Peter Winkler in 1988. It is easy to see that when H is a reflexive 3-cycle, *RET-H*, and hence *COMP-H*, are both polynomial time solvable. In fact, when H is a reflexive chordal graph (which includes a reflexive 3-cycle), the problem *RET-H* is polynomial time solvable [Feder and Hell, 1998], and hence *COMP-H* is also polynomial time solvable.

It is also known that *RET-H* is NP-complete when H is an irreflexive even k-cycle, for all even $k \geq 6$, cf. [Feder, Hell, and Huang, 1999], G. MacGillivray, 1988 (personal communication). It is shown in [Vikas, 1999, 2004] that *COMP-H* is NP-complete when H is an irreflexive even k-cycle, for all even $k \geq 6$. This result in [Vikas, 1999, 2004] also solves a long-standing problem that has been of interest to various people including Pavol Hell and Jaroslav Nesetril (personal communications). It is easily seen that when H is an irreflexive 4-cycle, *RET-H*, and hence *COMP-H*, are both polynomial time solvable. In fact, when H is a chordal bipartite graph (which includes an irreflexive 4-cycle), the problem *RET-H* is polynomial time solvable [Bandelt, Dahlmann, and Schutte, 1987], and hence *COMP-H* is also polynomial time solvable.

The case of irreflexive odd cycles is a special case of a more general result whereby *RET-H* and *COMP-H* are polynomially equivalent for every nonbipartite irreflexive graph H. Note that a graph G is homomorphic to a graph H if and only if the disjoint union $G \cup H$ retracts/compacts to H. Thus we have a polynomial transformation from *H-COL* to *RET-H* and *COMP-H*. The problem *H-COL* is NP-complete for any nonbipartite irreflexive graph H [Hell and Nesetril, 1990]. It follows that *RET-H* and *COMP-H* are also NP-complete for any nonbipartite irreflexive graph H; in particular, if H is an irreflexive odd k-cycle then *RET-H* and *COMP-H* are NP-complete, for all odd $k \geq 3$. Thus we conclude that *RET-H* and *COMP-H* both are NP-complete when H is an irreflexive k-cycle, for all $k \geq 3$, $k \neq 4$.

In section 2, we give a polynomial transformation from *COMP-H* to *RET-H* under Turing reduction. We shall need to refer to this transformation later in our proofs. In section 3, we study the relationship between compaction and retraction problems for reflexive graphs, and then study its consequences in relation to the constraint satisfaction problem. In section 4, we study the relationship between compaction and retraction problems for bipartite graphs, and its consequences in relation to the constraint satisfaction problem.

2. A polynomial transformation from compaction to retraction. We prove the following theorem in this section.

THEOREM 2.1. *For every fixed graph H, the problem COMP-H is polynomially transformable to the problem RET-H under Turing reduction.*

Proof. Let H be a fixed graph, and let a graph G be an instance of *COMP-H*. For a graph Z, let $E^*(Z)$ denote the set of all nonloop edges of Z, and let $I(Z)$ denote the set of all isolated vertices in Z. We note from the definition of compaction that if $c : G \to H$ is a compaction then part of the requirement for c is that for all $x \in I(H)$, there exists $v \in V(G)$ with $c(v) = x$, and for all $hh' \in E^*(H)$, there exists $gg' \in E^*(G)$ with $c(g) = h$ and $c(g') = h'$. Thus, there exists a set of $|E^*(H)|$ nonloop edges of G which cover all the nonloop edges of H under c, and there exists a set of $|I(H)|$ vertices in G which cover all the isolated vertices in H under c. In other words, there exists a subgraph Q of G induced by $V(Q')$, where Q' is a subgraph (not necessarily an induced subgraph) of G with $|E^*(Q')| = |E^*(H)|$ and $|I(Q')| = |I(H)|$, such that $c : Q \to H$ is a compaction.

In general, there may exist several such above described induced subgraphs Q of G such that Q compacts to H, regardless of whether or not G compacts to H. Further, if Q compacts to H then there may exist several compactions f of Q to H. The pair (Q, f) will be used to denote such a subgraph Q of G with a compaction $f : Q \to H$. Note that for each such subgraph Q of G, there may be more than one but a fixed number of such pairs (since H is fixed). Also, note that for such a subgraph Q of G, we have $|V(Q)| \leq 2 * |E^*(H)| + |I(H)|$. Thus, since H is fixed, all such subgraphs Q of G, and all compactions f of Q to H, i.e., all the pairs (Q, f), can be found in time polynomial in the size of G.

Let β denote the number of different possible pairs (Q, f). Clearly, β is a polynomial in the size of G. Consider the ith pair (Q, f) (under an arbitrary ordering of the pairs), $1 \leq i \leq \beta$. We define $L_i(q) = \{f(q)\}$ (i.e., $L_i(q)$ is a singleton containing the vertex $f(q)$ of H), for all $q \in V(Q)$, and $L_i(u) = V(H)$, for all $u \in V(G) - V(Q)$. Thus we obtain the lists $L_i(v) \subseteq V(H)$, for all $v \in V(G)$, $i = 1, 2, \ldots, \beta$ in polynomial time. Clearly, G compacts to H if and only if there exists an i such that there is a list-L_i-homomorphism of G to H, $1 \leq i \leq \beta$.

We construct a graph G_i from G and L_i, for all $i = 1, 2, \ldots, \beta$, as follows. We identify every pair of distinct vertices u and w in G if $L_i(u)$ and $L_i(w)$ are singletons with $L_i(u) = L_i(w)$, and we call the resultant graph after identifications, the graph G_i, for all $i = 1, 2, \ldots, \beta$. Note that, as a result of the identifications, G_i has a copy of H as an induced subgraph, except that possibly some of the loops, that may be present on some of the vertices of H, may not be present in the copy of H in G_i, in which case we add the missing loops to obtain an exact copy of H in G_i, for all $i = 1, 2, \ldots, \beta$. Since β is a polynomial, we have only polynomially many graphs $G_1, G_2, \ldots, G_\beta$. Thus we obtain the graphs $G_1, G_2, \ldots, G_\beta$ in polynomial time. Clearly, G compacts to H if and only if there exists an i such that G_i retracts to H, $1 \leq i \leq \beta$. Thus we have a polynomial transformation from *COMP-H* to *RET-H* under Turing reduction. □

It follows from Theorem 2.1 that for a fixed graph H, if *RET-H* is solvable in polynomial time then *COMP-H* is also solvable in polynomial time. We shall also refer later the transformation from *COMP-H* to *RET-H* given in the proof of Theorem 2.1.

3. Relationship between compaction and retraction problems for reflexive graphs. In this section, we prove the following theorem showing a very close relationship between compaction and retraction problems for reflexive graphs, and

then study its consequences in relation to the constraint satisfaction problem.

THEOREM 3.1. *For every reflexive graph H, there exists a reflexive graph H' such that RET-H is polynomially equivalent to COMP-H'.*

Proof. Let H be a reflexive graph. We construct in a fixed time, a reflexive graph H' from H such that the following two statements (a) and (b) hold:

(a) *RET-H* polynomially transforms to *COMP-H'*.

(b) *COMP-H'* polynomially transforms to *RET-H*.

The theorem then follows from (a) and (b).

We assume that H is connected. Later, we will make remarks for the case when H may be disconnected.

Suppose first that the diameter of H is ≤ 1. Clearly then H is a complete graph. Let G be any graph containing H as an induced subgraph, i.e., let G be an instance of *RET-H*. Clearly, G always retracts and hence compacts to H. Let H' also be the graph H. Trivially, G retracts to H if and only if G compacts to H. Hence statement (a) is trivially true. We know from Theorem 2.1 in section 2 that *COMP-H* polynomially transforms to *RET-H* under Turing reduction. Hence statement (b) is true. Thus the theorem holds for reflexive graphs H of diameter ≤ 1.

Now suppose that the diameter of H is > 1. The construction of H' is as follows. Let x be any vertex of H such that there exists a vertex at distance > 1 from x in H. There exists such a vertex x in H, as the diameter of H is > 1. Let the maximum distance from x to a vertex of H be s (thus $s > 1$). Let $d_H(x, h) = s_h$, for all $h \in V(H)$. For each vertex h of H, we add to H a path $P_h = p_1^h p_2^h \dots p_{s-s_h+1}^h$ containing $s - s_h + 1$ new vertices with p_1^h adjacent to h. We also add the reflexive edge $p_i^h p_i^h$, for all $i = 1, 2, \dots, s - s_h + 1$, with $h \in V(H)$. The resultant graph is our H', which is reflexive. Clearly, H' is constructed in a fixed time, as H is fixed. We shall denote the vertex $p_{s-s_h+1}^h$ by t_h and call it the *tip* of the path P_h, with $h \in V(H)$. Note that, among the paths added to H, P_x is the longest path (of length s) added to H, and P_y is a shortest path (of length 0 containing only one vertex) added to H, where $y \in V(H)$ is a furthest vertex from x in H, i.e., $d_H(x, y) = s$. See Figure 3.1, where we show the path P_h for an arbitrary vertex h of H. Also shown in the figure are paths P_x and P_y, where y is a vertex of H with $d_H(x, y) = s$. The graph H is denoted as an ellipse in the figure. The loops

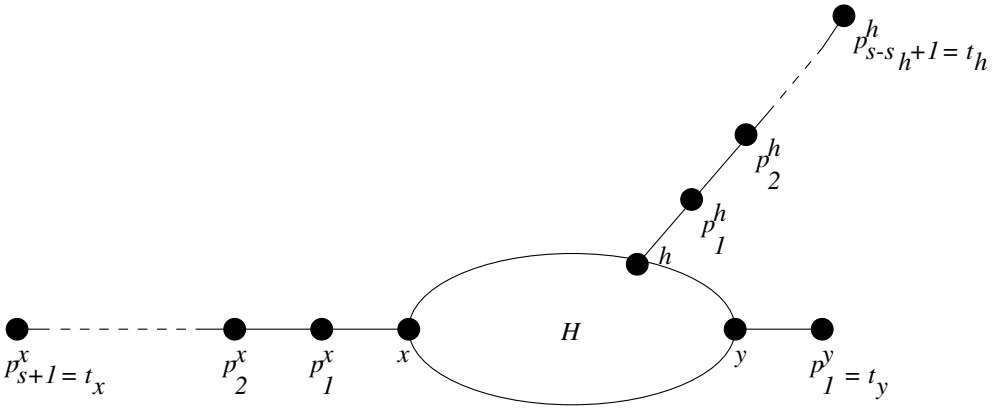


FIG. 3.1. *Construction of H'.*

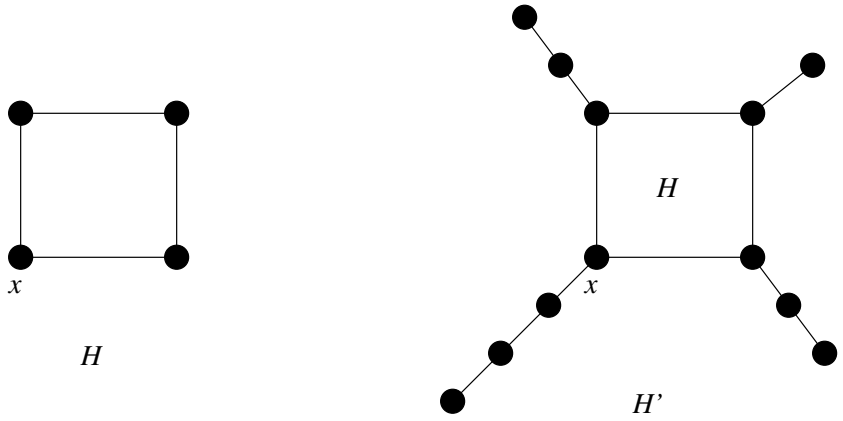


FIG. 3.2. *An example of H and H'.*

are not depicted in the figure but are assumed to be present on every vertex of H'. Clearly, H' retracts to H. See Figure 3.2 for an example of H and H'. Note that $d_{H'}(x,t_h) = d_{H'}(x,p^h_{s-s_h+1}) = d_{H'}(x,h)+d_{H'}(h,p^h_{s-s_h+1}) = s_h+(s-s_h+1) = s+1$, with $h \in V(H)$, and all other vertices are at distances less than $s+1$ from x in H'. Thus $d_{H'}(x,h) \leq s+1$, for all $h \in V(H')$, and the only vertices at distance $s+1$ from x in H' are the tip vertices t_h, for all $h \in V(H)$.

We prove the statements (a) and (b) in Theorem 3.1.1 and Theorem 3.1.2 respectively for the case when the diameter of H is > 1.

THEOREM 3.1.1. *RET-H polynomially transforms to COMP-H'.*

Proof. When the diameter of H is ≤ 1, we have already shown that the theorem is trivially proved. We assume here that the diameter of H is > 1. Let a graph G containing H as an induced subgraph be an instance of *RET-H*. We construct in time polynomial in the size of G, a graph G' (containing G and H' as induced subgraphs) such that the following statements (i) and (ii) are equivalent:

(i) G retracts to H.

(ii) G' compacts to H'.

This would imply that *RET-H* polynomially transforms to *COMP-H'*.

The construction of G' is as follows. We add to the subgraph H of G, the vertices of $V(H') - V(H)$ and the edges of $E(H') - E(H)$ such that H is expanded to the graph H' (the vertices of $H' - H$ are not adjacent to any vertex of $G - H$). For each vertex v of $G - H$, we add to G a path $U_v = u^v_1 u^v_2 \dots u^v_{s-1}$ containing $s-1$ new vertices, and we add the edges xu^v_1 and vu^v_{s-1}. This completes the construction of G'.

Note that $d_{G'}(x,v) \leq d_{G'}(x,u^v_1)+d_{G'}(u^v_1,u^v_{s-1})+d_{G'}(u^v_{s-1},v) = 1+(s-2)+1 = s$, for all $v \in V(G-H)$. Clearly then $d_{G'}(x,w) \leq s$, for all $v \in V(G') - V(H')$. Earlier we noted that $d_{H'}(x,h) \leq s+1$, for all $h \in V(H')$, and the only vertices at distance $s+1$ (calculated in H') from x in H' are the tip vertices t_h, for all $h \in V(H)$. Thus $d_{G'}(x,u) \leq s+1$, for all $u \in V(G')$, and the only vertices that are possibly at distance $s+1$ (calculated in G') from x in G' are the tip vertices t_h, for all $h \in V(H)$. We are saying $s+1$ to be an upper bound for $d_{G'}(x,t_h)$ also, as the presence of an edge from a vertex of $G-H$ to a vertex of H may result in a path of length shorter than $s+1$ from x to a tip vertex t_h in G', with $h \in V(H)$.

We now prove the equivalence of statements (i) and (ii), from which Theorem 3.1.1 follows.

First suppose that $r : G \to H$ is a retraction. We define below a retraction $r' : G' \to H'$, which by definition is also a compaction.

We define

$r'(v) = r(v)$, for all $v \in V(G)$,

$r'(h') = h'$, for all $h' \in V(H') - V(H)$.

We now fix a vertex $v \in V(G - H)$ for defining r' for the vertices of U_v. Let $P_{xv} = h_0 h_1 \dots h_j$ be a shortest path from $r(x)$ to $r(v)$ in H, with $r(x) = h_0$ and $r(v) = h_j$, $h_i \in V(H)$, for all $i = 0, 1, 2, \dots, j$. Clearly, the length of the path P_{xv} is $\leq s$, as $d_H(x, h) \leq s$, for all $h \in V(H)$. Thus $j \leq s$.

For the vertices of U_v, we define

$r'(u_i^v) = h_i$, for all $i = 1, 2, \dots, j-1$,

$r'(u_i^v) = h_{j-1}$, for all $i = j, j+1, \dots, s-1$.

It is easily verified that $r' : G' \to H'$ is a retraction, and hence a compaction.

Now suppose that $c : G' \to H'$ is a compaction. We shall prove that $c(h) \neq c(h')$, for all $h, h' \in V(H')$, $h \neq h'$. In order to prove this, we shall be proving a number of other claims. Note that this would imply that $c : H' \to H'$ is an automorphism. As we will see, we can easily define a retraction of G' to H' from this fact, and finally conclude that G retracts to H, proving the theorem.

We first show that $c(x) = x$. If $c(x) \neq x$ then either $c(x) \in V(P_x)$ or $c(x) \in V(H') - \{V(P_x) \cup \{x\}\}$. Since $s > 0$, there exists a vertex $h \in V(H)$, $h \neq x$. For every $p \in V(P_x)$, we have $d_{H'}(p, t_h) = d_{H'}(p, x) + d_{H'}(x, t_h) > d_{H'}(x, t_h) = s+1$. Since there is no vertex at distance greater than $s+1$ from x in G', this implies that $c(x) \neq p$, for all $p \in V(P_x)$, as otherwise no vertex of G' will map to t_h under c, contradicting that $c : G' \to H'$ is a compaction. Since $d_{H'}(z, t_x) = d_{H'}(z, x) + d_{H'}(x, t_x) > d_{H'}(x, t_x) = s + 1$, similarly we argue that $c(x) \neq z$, for all $z \in V(H') - \{V(P_x) \cup \{x\}\}$. Thus we have shown that $c(x) \notin V(P_x)$ and $c(x) \notin V(H') - \{V(P_x) \cup \{x\}\}$. Hence $c(x) = x$.

We next show that for each vertex t_h, with $h \in V(H)$, there exists an unique vertex t_{z_h}, with $z_h \in V(H)$, such that $c(t_{z_h}) = t_h$. That is, we show that the tips are mapped to tips under c, and exactly one tip maps to a tip under c. We have $d_{H'}(c(x) = x, t_h) = s+1$, for all $h \in V(H)$. Also, $d_{G'}(x, u) \leq s+1$, for all $u \in V(G')$, and the only vertices that can possibly be at distance $s+1$ from x in G' are t_h, for all $h \in V(H)$. This implies that for each vertex t_h, with $h \in V(H)$, there exists a unique vertex t_{z_h}, with $z_h \in V(H)$, such that $c(t_{z_h}) = t_h$. Thus every tip is mapped to a tip under c, and exactly one tip maps to a tip under c.

Now we show that $d_{H'}(x, h') = d_{H'}(c(x), c(h'))$, for all $h' \in V(H')$, and $d_{H'}(h, t_h) = d_{H'}(c(h), c(t_h))$, for all $h \in V(H)$. It follows from the above result that $d_{H'}(c(x), c(t_h)) = s+1$, for all $h \in V(H)$. It must be that $d_{H'}(c(h), c(h')) \leq d_{H'}(h, h')$, for all $h, h' \in V(H')$, as $c : G' \to H'$ is a homomorphism. Clearly, $d_{H'}(c(x), c(t_h)) \leq d_{H'}(c(x), c(h')) + d_{H'}(c(h'), c(t_h)) \leq d_{H'}(x, h') + d_{H'}(h', t_h) = d_{H'}(x, t_h) = s + 1$, for all $h' \in \{h\} \cup V(P_h)$, $h \in V(H)$. Thus if $d_{H'}(c(x), c(h')) < d_{H'}(x, h')$ or if $d_{H'}(c(h'), c(t_h)) < d_{H'}(h', t_h)$ then we will have that $d_{H'}(c(x), c(t_h)) < s + 1$, which would be a contradiction, with $h' \in \{h\} \cup V(P_h)$, $h \in V(H)$. Thus $d_{H'}(c(x), c(h')) = d_{H'}(x, h')$, and $d_{H'}(c(h'), c(t_h)) = d_{H'}(h', t_h)$, for all $h' \in \{h\} \cup V(P_h)$, $h \in V(H)$. This implies that $d_{H'}(x, h') = d_{H'}(c(x), c(h'))$, for all $h' \in V(H')$, and $d_{H'}(h, t_h) = d_{H'}(c(h), c(t_h))$, for all $h \in V(H)$.

We now prove a few claims below, concluding that we can choose the compaction $c : G' \to H'$ in such a way that $c(h) \in V(H)$, for all $h \in V(H)$.

We show that for each path P_h, with $h \in V(H)$, there exists at most one vertex $z \in V(H)$ such that $c(z) \in V(P_h)$, and if $c(z) \in V(P_h)$ then $c(t_z) = t_h$. Suppose that

there exist two distinct vertices $z, z' \in V(H)$ such that $c(z), c(z') \in V(P_h)$, for some $h \in V(H)$. We have $d_{H'}(c(x), p) + d_{H'}(p, t_y) = d_{H'}(c(x), h) + d_{H'}(h, p) + d_{H'}(p, h) + d_{H'}(h, t_y) > d_{H'}(c(x), h) + d_{H'}(h, t_y) \geq d_{H'}(c(x), t_y) = s + 1$, for all $p \in V(P_h)$, $y \in V(H)$, $y \neq h$. We have $d_{H'}(x, t_z) = d_{H'}(x, z) + d_{H'}(z, t_z) = s + 1$. Under the assumption that $c(z) \in V(P_h)$, this implies that $d_{H'}(x, z) + d_{H'}(z, t_z) < d_{H'}(c(x), c(z)) + d_{H'}(c(z), t_y)$, for all $y \in V(H)$, $y \neq h$. Since $d_{H'}(x, z) = d_{H'}(c(x), c(z))$ (as proved earlier), we have that $d_{H'}(z, t_z) < d_{H'}(c(z), t_y)$, implying that $c(t_z) \neq t_y$, for all $y \in V(H)$, $y \neq h$. Similarly, $c(t'_z) \neq t_y$, for all $y \in V(H)$, $y \neq h$. Since tips map only to tips under c, this would imply that $c(t_z) = c(t_{z'}) = t_h$, which is a contradiction, as we showed earlier that exactly one tip maps to a tip under c. Thus there cannot exist more than one vertex of H with image in $V(P_h)$ under c, and if there exists such a vertex z of H with $c(z) \in V(P_h)$ then $c(t_z) = t_h$.

We also show that $c(z) \notin V(P_h) - \{p_1^h\}$, for all $z, h \in V(H)$. Suppose that $c(z) \in V(P_h) - \{p_1^h\}$, for some $z, h \in V(H)$. Clearly, x is not adjacent to z in H, as $c(x) \in V(H)$ and $c(z)$ is not adjacent to $V(H)$ as $c(z) \neq p_1^h$. Since H is connected and x is not adjacent to z, there exists a path from x to z of length ≥ 2 in H, and hence there exists a neighbor z' of z in H, $z' \neq z$. Since z' is adjacent to z in H, and $c(z) \in V(P_h) - \{p_1^h\}$, this implies that $c(z') \in V(P_h)$. Thus we have two distinct vertices z and z' of H which have images in $V(P_h)$ under c. This is impossible, as shown above.

Further, we show that $c(z) \notin V(P_h)$ if z has a neighbor z' in H, $z \neq z'$, such that $d_{H'}(x, z') \geq d_{H'}(x, z)$, for all $z, h \in V(H)$. Suppose that $c(z) \in V(P_h)$, for some $z, h \in V(H)$, and z has such a neighbor z' in H. We proved earlier that $d_{H'}(x, h') = d_{H'}(c(x), c(h'))$, for all $h' \in V(H')$. Hence we have $d_{H'}(x, z) = d_{H'}(c(x), c(z))$ and $d_{H'}(x, z') = d_{H'}(c(x), c(z'))$. Since $d_{H'}(x, z') \geq d_{H'}(x, z)$, this implies that $d_{H'}(c(x), c(z')) \geq d_{H'}(c(x), c(z))$. Since z' is adjacent to z, either $c(z') = h$ or $c(z') \in V(P_h)$. We shall prove that $c(z') \neq h$. Suppose that $c(z') = h$. We have $d_{H'}(c(x), c(z')) = d_{H'}(x, h)$ and $d_{H'}(c(x), c(z)) = d_{H'}(x, c(z)) = d_{H'}(x, h) + d_{H'}(h, c(z))$. Thus $d_{H'}(c(x), c(z')) < d_{H'}(c(x), c(z))$, which is a contradiction. Thus $c(z') \neq h$, and hence $c(z') \in V(P_h)$. We then have two distinct vertices z and z' of H which have images in $V(P_h)$ under c. This is impossible, as shown earlier.

Thus if a vertex z of H has an image in a path P_h under c, with $h \in V(H)$, then $c(z) = p_1^h$ and $d_{H'}(x, z') < d_{H'}(x, z)$, for any neighbor z' of z in H. Suppose that $c(z) = p_1^h$, for some $z, h \in V(H)$. Let z' be any neighbor of z in H, $z' \neq z$ (since H is connected and clearly $z \neq x$, there exists such a neighbor). We proved earlier that there cannot exist more than one vertex of H with image in $V(P_h)$ under c. Since $c(z) = p_1^h \in V(P_h)$ and z' is a neighbor of z in H, this implies that $c(z') = h$. As proved earlier, since $c(z) \in V(P_h)$, we have $c(t_z) = t_h$. Also, as proved earlier, we have $d_{H'}(z, t_z) = d_{H'}(c(z) = p_1^h, c(t_z) = t_h)$, and hence $c(p_i^z) = p_{i+1}^h$, for all $i = 1, 2, \ldots, s - s_z + 1 = s - s_h$ (recall that $p_{s-s_z+1}^z = t_z$ and $p_{s-s_h+1}^h = t_h$). Further, since exactly one tip maps to a tip under c, this implies that $c(t_{z'}) = t_y$, for some $y \in V(H)$, $y \neq h$. Note, it must be that the length of the path P_y is smaller than the length of the path P_h (length of P_h = length of $P_{z'}$, as $d_{H'}(x, z') = d_{H'}(c(x) = x, c(z') = h)$), i.e., $d_{H'}(x, y) > d_{H'}(x, h) (= d_{H'}(x, z'))$, in order that $c(t_{z'}) = t_y$ (as some of the vertices of $P_{z'}$ are now mapped to H under c, as $d_{H'}(z', t_{z'}) = d_{H'}(c(z') = h, c(t_{z'}) = t_y)$; clearly at least $c(p_1^{z'}) \in V(H)$). We know that $d_{H'}(x, z') < d_{H'}(x, z)$, and hence $d_{H'}(x, z') = d_{H'}(x, z) - 1$. Thus the length of the path P_z is 1 less than the length of the path $P_{z'}$. Let $S = \{v | v \in V(G') - V(H') \text{ and } c(v) \in V(P_h)\}$. Note that apart from the vertices of $\{z\} \cup P_z$, only vertices in S may map to a vertex of P_h under c.

We can define a compaction $c' : G' \to H'$ such that $c'(z) \in V(H)$ by interchanging $c(t_z)$ and $c(t_{z'})$ as follows:

$c'(v) = c(v)$, for all $v \in V(G') - \{V(P_z) \cup V(P_{z'}) \cup \{z\} \cup S\}$,
$c'(v) = h$, for all $v \in S$,
$c'(p_i^{z'}) = p_i^h$, for all $i = 1, 2, \ldots, s - s_{z'} + 1 = s - s_h + 1$
(thus $c'(t_{z'}) = t_h = c(t_z)$ and $c(p_1^{z'}) = p_1^h = c(z)$),
$c'(z) = c(p_1^{z'})$ and $c'(p_i^z) = c(p_{i+1}^{z'})$, for all $i = 1, 2, \ldots, s - s_z + 1 = s - s_{z'}$
(thus $c'(z) \in V(H)$, as $c(p_1^{z'}) \in V(H)$, and $c'(t_z) = c(t_{z'})$).

It can be easily verified that $c' : G' \to H'$ is a compaction. We have $c'(z) \in V(H)$. We apply similar arguments for any path $P_{h'}$ of H', with $h' \in V(H)$, to which a vertex of H may map under c. This implies that there exists a compaction of G' to H' such that all the vertices of H have images in H under this compaction. We choose $c : G' \to H'$ to be such a compaction.

We now show that for any vertex z of H, if $c(z) = h$, for some vertex h of H (we assume $h \in V(H)$, as we proved that we choose c such that the vertices of H map to H under c), then $c(t_z) = t_h$.

Let $c(z_0) = h_0$, for some $z_0, h_0 \in V(H)$. Suppose that $c(t_{z_0}) \neq t_{h_0}$. Since tips map to tips under c, this implies that $c(t_{z_0}) = t_{h_1}$, for some vertex $h_1 \in V(H)$, $h_1 \neq h_0$. Note that the length of the path P_{h_1} is necessarily smaller than the length of the path P_{z_0}, as some of the vertices of P_{z_0} now map to H under c; clearly at least $c(p_1^{z_0}) \in V(H)$. From the construction of H', this implies that $d_{H'}(x, h_1) > d_{H'}(x, z_0)$. Since $d_{H'}(x, h) = d_{H'}(c(x), c(h))$, for all $h \in V(H')$, we have $d_{H'}(x, z_0) = d_{H'}(c(x) = x, c(z_0) = h_0)$. Thus $d_{H'}(x, h_1) > d_{H'}(x, h_0)$.

We show that there exists a vertex $z_1 \in Z = \{z | z \in V(H)$ and $d_{H'}(x, z) = d_{H'}(x, h_1)\}$ such that $c(t_{z_1}) \neq t_z$, for all $z \in Z$. Since $d_{H'}(x, h) = d_{H'}(c(x) = x, c(h))$ and $c(h) \in V(H)$, for all $h \in V(H)$, this implies that $c(z) \in Z$, for all $z \in Z$. First suppose that $c(y) = h_1$, for some vertex $y \in Z$ (trivially, $h_1 \in Z$). Then $c(t_y) \neq t_{h_1}$, as $c(t_{z_0}) = t_{h_1}$ and exactly one tip maps to a tip under c. Note that for any pair of vertices a and b in Z, the length of the path P_a is the same as the length of the path P_b. Hence $d_{H'}(y, t_y) = d_{H'}(z, t_z) < d_{H'}(h_1, z) + d_{H'}(z, t_z) = d_{H'}(c(y) = h_1, t_z)$, for all $z \in Z$, $z \neq h_1$. This implies that $c(t_y) \neq t_z$, for all $z \in Z$, $z \neq h_1$. Since $c(t_y) \neq t_{h_1}$, we have $c(t_y) \neq t_z$, for all $z \in Z$.

Now suppose that $c(z) \neq h_1$, for all $z \in Z$. Then clearly there exists a pair of distinct vertices y and y' in Z such that $c(y) = c(y') = u$, for some $u \in Z$. Since exactly one tip maps to a tip under c, we have $c(t_y) \neq t_u$ or $c(t_{y'}) \neq t_u$ (both may hold). Without loss of generality, suppose that $c(t_y) \neq t_u$. We have $d_{H'}(y, t_y) = d_{H'}(z, t_z) < d_{H'}(u, z) + d_{H'}(z, t_z) = d_{H'}(c(y) = u, t_z)$, for all $z \in Z$, $z \neq u$. This implies that $c(t_y) \neq t_z$, for all $z \in Z$, $z \neq u$. Since $c(t_y) \neq t_u$, we have $c(t_y) \neq t_z$, for all $z \in Z$. Earlier we showed above that if $c(y) = h_1$, for any vertex $y \in Z$, then $c(t_y) \neq t_z$, for all $z \in Z$. Thus we have proved that there exists a vertex $z_1 \in Z$ such that $c(t_{z_1}) \neq t_z$, for all $z \in Z$. Since tips map to tips under c, this implies that $c(t_{z_1}) = t_{h_2}$, for some vertex $h_2 \in V(H)$, $h_2 \notin Z$, such that the length of the path P_{h_2} is smaller than the length of the path P_{z_1} (as some of the vertices of P_{z_1} now map to H under c; clearly at least $c(p_1^{z_1}) \in V(H)$), i.e., $d_{H'}(x, h_2) > d_{H'}(x, z_1) = d_{H'}(x, h_1)$. For convenience, we call Z the set Z_1, and we define $Z_0 = \{z | z \in V(H)$ and $d_{H'}(x, z) = d_{H'}(x, h_0)\}$. Thus $z_0 \in Z_0$ and $z_1 \in Z_1$.

Earlier we showed that $c(t_{z_0}) = t_{h_1}$ and $d_{H'}(x, h_1) > d_{H'}(x, h_0)$, and now we have shown that $c(t_{z_1}) = t_{h_2}$ and $d_{H'}(x, h_2) > d_{H'}(x, h_1)$. We can continue this argument and show that there exists a vertex $h_{i+1} \in V(H)$ and a vertex $z_i \in Z_i = \{z | z \in V(H)$

and $d_{H'}(x,z) = d_{H'}(x,h_i)\}$ such that $c(t_{z_i}) = t_{h_{i+1}}$ and $d_{H'}(x,h_{i+1}) > d_{H'}(x,h_i)$, for all $i = 0,1,2,\ldots,p$, where $d_{H'}(x,h_{p+1}) > s$. This is a contradiction, as no vertex of H is at distance larger than s from x in H'. Thus our assumption that $c(t_{z_0}) \neq t_{h_0}$ is wrong. Thus for any vertex z of H, if $c(z) = h$, for some vertex h of H, then $c(t_z) = t_h$.

Now suppose that $c(y) = c(z) = h$, for some $y,z,h \in V(H)$, $y \neq z$. From the above result, this implies that $c(t_y) = c(t_z) = t_h$. This is a contradiction, as exactly one tip maps to a tip under c. Hence $c(y) \neq c(z)$, for all $y,z \in V(H)$, $y \neq z$. Let $c(z) = h$, for some $z,h \in V(H)$. Since $d_{H'}(x,z) = d_{H'}(c(x) = x, c(z) = h)$, it follows from our construction that the length of the path P_z is same as the length of the path P_h. Since, as shown above, $c(t_z) = t_h$, we have $c(p_i^z) = p_i^h$, for all $i = 1,2,\ldots,s - s_z + 1 = s - s_h + 1$ (recall that $p_{s-s_z+1}^z = t_z$ and $p_{s-s_h+1}^h = t_h$). This completes the proof that $c(h) \neq c(h')$, for all $h,h' \in V(H')$, $h \neq h'$.

Since $c(h) \neq c(h')$, for all $h,h' \in V(H')$, $h \neq h'$, this implies that $c : H' \to H'$ is an automorphism. Let $f : H' \to H'$ be the automorphism $c : H' \to H'$. We define a retraction $r : G' \to H'$ as follows:

$$r(a) = f^{-1}(c(a)), \text{ for all } a \in V(G').$$

Thus G' retracts to H'. Since H' retracts to H, it follows that G' retracts to H. Hence G retracts to H, as G is a subgraph of G'. We have thus proved that (i) is equivalent to (ii), and the theorem follows. □

THEOREM 3.1.2. *COMP-H′ polynomially transforms to RET-H.*

Proof. We have already shown that the theorem holds when the diameter of H is ≤ 1. We assume here that the diameter of H is > 1. Let a graph G be an instance of *COMP-H′*. We construct in time polynomial in the size of G, polynomial (in the size of G) number of graphs $G_1, G_2, \ldots, G_\beta$, each containing a copy of H' as an induced subgraph, and for each G_i, we construct in time polynomial (in the size of G_i and hence in the size of G), a graph G_i' also containing a copy of H' as an induced subgraph, $1 \leq i \leq \beta$, such that the following statements (i), (ii), (iii), and (iv) are equivalent for some value of i, $1 \leq i \leq \beta$:

(i) G compacts to H'.
(ii) G_i retracts to H'.
(iii) G_i' retracts to H'.
(iv) G_i' retracts to H.

We prove that (i) is equivalent to (ii), (ii) is equivalent to (iii), and (iii) is equivalent to (iv). Thus, in effect, we prove that (i) is equivalent to (iv), which shows that *COMP-H′* polynomially transforms to *RET-H* under Turing reduction.

The graphs $G_1, G_2, \ldots, G_\beta$ are constructed as in the proof of Theorem 2.1 in section 2 (where we replace H by H'). As discussed there, β is a polynomial in the size of G, and the graphs $G_1, G_2, \ldots, G_\beta$ are constructed in time polynomial in the size of G. The equivalence of (i) and (ii) follows from the proof of Theorem 2.1.

For constructing G_i', we go through a number of steps that follow. We define $LST_i(h) = \{h\}$, for all $h \in V(H')$, and $LST_i(v) = V(H')$, for all $v \in V(G_i) - V(H')$, where H' is the copy in G_i, for all $i = 1,2,\ldots,\beta$. We perform the consistency test for G_i with respect to H' and LST_i, and obtain $L_i(v) \subseteq LST_i(v) \subseteq V(H')$, for all $v \in V(G_i)$, after completing the consistency test (recall that the consistency test will run in time linear in the size of G_i), for all $i = 1,2,\ldots,\beta$. It is clear that every possible retraction $r_i : G_i \to H'$ is a list-L_i-homomorphism, and every possible list-L_i-homomorphism $l_i : G_i \to H'$ is a retraction, for all $i = 1,2,\ldots,\beta$. Thus G_i retracts to H' if and only if there exists a list-L_i-homomorphism of G_i to H', for all $i = 1,2,\ldots,\beta$.

Suppose that G_i retracts to H', i.e., there exists a list-L_i-homomorphism of G_i to H', for some value of i, $1 \leq i \leq \beta$ (hence $L_i(v) \neq \phi$, for all $v \in V(G_i)$, i.e., the consistency test we performed for G_i has succeeded). Let $r : G_i \rightarrow H'$ be a retraction (thus $r : G_i \rightarrow H'$ is a list-L_i-homomorphism). We define another retraction $r' : G_i \rightarrow H'$ below. First we partition the vertex set of G_i into two sets V_1 and V_2 as follows:

$V_1 = \{v \in V(G_i) | L_i(v) \subseteq \{h_v\} \cup V(P_{h_v}),$ for some vertex $h_v \in V(H)\}$,
$V_2 = V(G_i) - V_1$.

For $v \in V_1$, we let

$r'(v) = h,$ where h is the vertex in $L_i(v)$ closest to h_v, that is, $d_{H'}(h, h_v) < d_{H'}(h', h_v),$ for all $h' \in L_i(v), h' \neq h$.

For $v \in V_2$, we let

$r'(v) = h,$ if $r(v) \in \{h\} \cup V(P_h),$ with $h \in V(H)$.

This completes the definition of r'.

Since $L_i(h) = \{h\}$, for all $h \in V(H')$, we have $V(H') \subseteq V_1$ and it is clear from the definition of r' that $r'(h) = h$, for all $h \in V(H')$. We now verify that $r' : G_i \rightarrow H'$ is indeed a homomorphism (and hence a retraction). We shall do this by considering all the edges ab of G_i and proving that $r'(a)r'(b)$ is an edge of H'.

Consider first an edge ab of G_i, with $a, b \in V_1$. From the definition of V_1, $L_i(a) \subseteq \{h_a\} \cup V(P_{h_a})$ and $L_i(b) \subseteq \{h_b\} \cup V(P_{h_b})$ (h_a may be the same as h_b), for some $h_a, h_b \in V(H)$. Let the vertex in $L_i(a)$ closest to h_a be h'_a, and the vertex in $L_i(b)$ closest to h_b be h'_b, i.e., $r'(a) = h'_a$ and $r'(b) = h'_b$. If $h_a \neq h_b$ then clearly $L_i(a) = \{h_a\}$ and $L_i(b) = \{h_b\}$, and h_a is adjacent to h_b in H, as L_i is nonempty and these are the only vertices the consistency test will keep. Thus if $h_a \neq h_b$ then $r'(a)r'(b) = h'_a h'_b = h_a h_b$ is an edge of H'. Now suppose that $h_a = h_b$. If $d_{H'}(h'_a, h_a) = d_{H'}(h'_b, h_a)$ then clearly $h'_a = h'_b$ and $r'(a)r'(b) = h'_a h'_b = h'_a h'_a$ is an edge of H'. Now assume that $d_{H'}(h'_a, h_a) < d_{H'}(h'_b, h_a)$. If h'_a is not adjacent to h'_b in H' then this implies that h'_b is not the closest vertex to h_a in $L_i(b)$, as otherwise the consistency test would have deleted h'_a from $L_i(a)$. Thus h'_a is adjacent to h'_b in H'. Similarly, we argue that if $d_{H'}(h'_a, h_a) > d_{H'}(h'_b, h_a)$ and h'_a is not adjacent to h'_b in H' then h'_a is not the closest vertex to h_a in $L_i(a)$, and hence h'_a is adjacent to h'_b in H'. Thus $r'(a)r'(b) = h'_a h'_b$ is always an edge of H'.

Next consider an edge ab of G_i, with $a, b \in V_2$. Let $r(a) \in \{h\} \cup V(P_h)$, for some $h \in V(H)$. Since $r(a)r(b)$ is an edge of H', this implies that $r(b) \in \{h\} \cup V(P_h)$ or $r(b) = h'$, for some h' adjacent to h, $h' \in V(H)$, $h' \neq h$. Hence, from the definition of r', we have $r'(a)r'(b) = hh$ or hh'. Thus $r'(a)r'(b)$ is always an edge of H' (and of H).

Now consider an edge ab of G_i, with $a \in V_1$ and $b \in V_2$. From the definition of V_2, $L_i(b) \not\subseteq \{h\} \cup V(P_h)$, for all $h \in V(H)$. Hence there exist two vertices h_1 and h_2 in $L_i(b)$, with $h_1 \in \{h\} \cup V(P_h)$ and $h_2 \in \{h'\} \cup V(P_{h'})$, for some $h, h' \in V(H)$, $h \neq h'$ (thus $h_1 \neq h_2$). From the definition of V_1, $L_i(a) \subseteq \{h_a\} \cup V(P_{h_a})$, for some vertex $h_a \in V(H)$. Since the consistency test we performed for G_i has succeeded, if $h_j \notin \{h_a\} \cup V(P_{h_a})$ then $h_j \in V(H) - \{h_a\}$ and is adjacent to h_a, and $h_a \in L_i(a)$, $j = 1, 2$. We know that at least one of $h_1, h_2 \notin \{h_a\} \cup V(P_{h_a})$, as at least one of h, h' is different from h_a. It follows that $h_a \in L_i(a)$, and from the definition of r', we have $r'(a) = h_a$. Now first suppose that $r(b) \notin \{h_a\} \cup V(P_{h_a})$. This implies that $r(a) = h_a$ and $r(b) = h_b$, for some vertex $h_b \in V(H)$ adjacent to h_a in H', $h_b \neq h_a$ (since $r(a) \in L_i(a) \subseteq \{h_a\} \cup V(P_{h_a})$ and $r : G_i \rightarrow H'$ is a homomorphism). Thus, from the definition of r', we have $r'(b) = h_b$. Hence $r'(a)r'(b) = h_a h_b$ is an edge of

H'. Now suppose that $r(b) \in \{h_a\} \cup V(P_{h_a})$. Then from the definition of r', we have $r'(b) = h_a$, and hence $r'(a)r'(b) = h_a h_a$ is an edge of H'.

Thus we have proved that $r' : G_i \to H'$ is a homomorphism, and hence a retraction. We now define a new set of lists L'_i as follows:

$L'_i(v) = \{r'(v)\}$, for all $v \in V_1$,

$L'_i(v) = L_i(v)$, for all $v \in V_2$.

Thus $L'_i(v) \subseteq L_i(v)$, for all $v \in V(G_i)$. Since every retraction of G_i to H' is a list-L_i-homomorphism of G_i to H', it follows that $r' : G_i \to H'$ is a list-L_i-homomorphism and a list-L'_i-homomorphism. Thus for every retraction $r : G_i \to H'$, there exists a list-L'_i-homomorphism $r' : G_i \to H'$. Clearly, every list-L'_i-homomorphism $l'_i : G_i \to H'$ is a retraction. Hence G_i retracts to H' if and only if there exists a list-L'_i-homomorphism of G_i to H'.

We now identify every pair of vertices v and v' in G_i for which $L'_i(v) = L'_i(v')$, $v, v' \in V_1$ (we know that $L'_i(a)$ is a singleton, for all $a \in V_1$), and call the resultant graph G'_i. Recall that for every vertex $h \in V(H')$, we have $L_i(h) = \{h\}$, and hence $h \in V_1$, and thus $L'_i(h) = \{r'(h) = h\}$. Clearly, G'_i retracts to H' if and only if there exists a list-L'_i-homomorphism of G_i to H'. Hence G_i retracts to H' if and only if G'_i retracts to H'. Thus we have proved that (ii) is equivalent to (iii). Note that we could have very well chosen $L'_i(v) = L_i(v) \cap V(H)$, for all $v \in V_2$, and all the above discussions would still hold, as $r'(v) \in V(H)$, for all $v \in V_2$.

We look at the structure of G'_i. Only vertices in V_1 have been identified with vertices in H'. For corresponding vertices of G_i and G'_i, we call them the same name. Thus $V(G'_i) = V(H') \cup V_2$. Note that no vertex v of $V(G'_i) - V(H') = V_2$ is adjacent to any vertex of P_h in G'_i, for all $h \in V(H)$. To see this, we recall our earlier explanation that if a vertex v of V_2 is adjacent to a vertex v' of V_1 in G_i then $h_{v'} \in L_i(v') \subseteq \{h_{v'}\} \cup V(P_{h_{v'}})$, with $h_{v'} \in V(H)$ (due to success of the consistency test). Hence $L'_i(v') = \{r'(v')\} = \{h_{v'}\}$, and in constructing G'_i, v' is identified with $h_{v'}$ which belongs to $V(H)$. Hence no vertex v of $V(G'_i) - V(H') = V_2$ can possibly be adjacent to any vertex of P_h in G'_i, for all $h \in V(H)$. Since G'_i contains H' as an induced subgraph and no vertex of $V(G'_i) - V(H')$ is adjacent to any vertex of P_h, for all $h \in V(H)$, it follows that G'_i retracts to H' if and only if G'_i retracts to H. Thus we have proved that (iii) is equivalent to (iv).

Thus we have shown that G compacts to H' if and only if G'_i retracts to H for some value of i, $1 \leq i \leq \beta$, which shows that *COMP-H'* polynomially transforms to *RET-H* under Turing reduction. □

We have thus proved Theorem 3.1 when H is connected. Now suppose that H may be disconnected. Let $H_1, H_2, \ldots, H_p$ be the components of H, $p \geq 1$. For each component H_i, we construct a graph H'_i analogous to the construction described earlier for the connected case depending on whether the diameter of H_i is ≤ 1 or > 1, $1 \leq i \leq p$. For a graph H_i with diameter > 1, we may assume x_i and s_i to play the role of x and s respectively described earlier for the connected case, $1 \leq i \leq p$. For a graph H_i with diameter ≤ 1, we let x_i be any vertex of H_i, and s_i (≤ 1) is defined similarly, $1 \leq i \leq p$. Our graph H' is the set of components $H'_1, H'_2, \ldots, H'_p$. We have to prove statements (a) and (b) mentioned at the beginning of the proof of Theorem 3.1.

For proving statement (a), we need to consider the construction of the graph G' in the proof of Theorem 3.1.1. Let $G_1, G_2, \ldots, G_t$ be the components of a graph G, with H_i as an induced subgraph of G_i, for all $i = 1, 2, \ldots, p$. Thus G is an instance of *RET-H*. If the diameter of H_i is > 1, the construction of G'_i is analogous to the

construction described in the proof of Theorem 3.1.1, $1 \leq i \leq p$. If the diameter of H_i is ≤ 1 then G'_i is obtained simply by adding an edge $x_i v$ in G_i, for each vertex v of $G_i - H_i$; thus the diameter of G'_i is ≤ 2, $1 \leq i \leq p$. When constructing G'_1, we also add an edge $x_1 v$ in G_1, for each vertex v of $G_{p+1}, G_{p+2}, \ldots, G_t$. Note that the diameter of G'_1 is still ≤ 2 if the diameter of H_1 is ≤ 1. Our graph G' is the set of components $G'_1, G'_2, \ldots, G'_p$.

If G retracts to H then it can be seen that G' retracts to H', and hence G' compacts to H'. Now suppose that G' compacts to H', and let $c : G' \to H'$ be a compaction. No two components of G' compact to the same component of H' under c, as the number of components in G' and H' is the same. Without loss of generality, suppose that the diameter of H_i is > 1, for all $i = 1, 2, \ldots, j$, and is ≤ 1, for all $i = j+1, j+2, \ldots, p$, $0 \leq j \leq p$. Since the diameter of H'_i is ≥ 3, for all $i = 1, 2, \ldots, j$, and the diameter of G'_k is ≤ 2, for all $k = j+1, j+2, \ldots, p$, no component among $G'_{j+1}, G'_{j+2}, \ldots, G'_p$ compacts to any component among $H'_1, H'_2, \ldots, H'_j$. Thus only the components among $G'_1, G'_2, \ldots, G'_j$ compact to the components among $H'_1, H'_2, \ldots, H'_j$. We know that H'_i is an induced subgraph of G'_i, for all $i = 1, 2, \ldots, j$. We note from our constructions that if G'_i compacts to H'_m under c then H'_m must be isomorphic to H'_i, as otherwise we will have a tip of some component H'_n not covered under c, $1 \leq i \leq j$, $1 \leq m \leq j$, $1 \leq n \leq j$. Thus we conclude that G'_i compacts to H'_i, for all $i = 1, 2, \ldots, j$. Since H_i is connected, we prove as in the proof of Theorem 3.1.1 that G_i retracts to H_i, for all $i = 1, 2, \ldots, j$. Clearly, G'_k and G_k retract and hence compact to H'_k, as H'_k, which is the same graph as H_k, is a complete graph, for all $k = j+1, j+2, \ldots, p$. Also, $G_{p+1}, G_{p+2}, \ldots, G_t$ is homomorphic to H, as H is reflexive. Thus we have that G retracts to H. Hence statement (a) follows.

For proving statement (b), we just need to assume in the proof of Theorem 3.1.2 that if the diameter of H_i is ≤ 1 then $V(P_h) = \phi$, for all $h \in V(H_i)$, $1 \leq i \leq p$. □

3.1. Compaction to reflexive graphs in relation to constraint satisfaction. It is shown in [Feder and Vardi, 1998] that for every fixed template T of a set of relations, there exists a bipartite graph H such that the constraint satisfaction problem *CSP-T* is polynomially equivalent to the retraction problem *RET-H*. Since it is thought to be likely difficult to determine whether for every template T, the problem *CSP-T* is polynomial time solvable or NP-complete, it follows that it is equally difficult to determine whether for every bipartite graph H, the problem *RET-H* is polynomial time solvable or NP-complete.

It has been shown in [Feder and Hell, 1998] that for every bipartite graph H, there exists a reflexive graph H' such that *RET-H* is polynomially equivalent to *RET-H'*. It follows from this result of [Feder and Hell, 1998] and the above result of [Feder and Vardi, 1998] that for every template T of a set of relations, there exists a reflexive graph H such that the constraint satisfaction problem *CSP-T* is polynomially equivalent to the retraction problem *RET-H*. This is taken as evidence in [Feder and Hell, 1998] that determining whether for every reflexive graph H, the problem *RET-H* is polynomial time solvable or NP-complete is likely to be difficult also.

We have shown in Theorem 3.1 that for every reflexive graph H, there exists a reflexive graph H' such that *RET-H* is polynomially equivalent to *COMP-H'*. Thus we have the following theorem which follows from Theorem 3.1 and the above mentioned results of [Feder and Vardi, 1998] and [Feder and Hell, 1998].

THEOREM 3.2. *For every fixed template T of a set of relations, there exists a fixed reflexive graph H such that the constraint satisfaction problem CSP-T is polynomially equivalent to the compaction problem COMP-H.* □

Thus we have evidence, due to Theorem 3.2, that it is likely to be difficult to determine whether for every reflexive graph H, the problem *COMP-H* is polynomial time solvable or NP-complete.

4. Relationship between compaction and retraction problems for bipartite graphs. In this section, we prove the following theorem showing a very close relationship between compaction and retraction problems for bipartite graphs, and then study its consequences in relation to the constraint satisfaction problem.

THEOREM 4.1. *For every bipartite graph H, there exists a bipartite graph H' such that RET-H is polynomially equivalent to COMP-H′.*

Proof. The proof is very similar to the proof of Theorem 3.1 for reflexive graphs. Here we will only highlight the differences in the proofs.

Let H be a bipartite graph. We construct in a fixed time, a bipartite graph H' from H such that the following two statements (a) and (b) hold:

(a) *RET-H* polynomially transforms to *COMP-H′*.

(b) *COMP-H′* polynomially transforms to *RET-H*.

The theorem then follows from (a) and (b).

We assume that H is connected. Later, we will make remarks for the case when H may be disconnected.

We had shown how Theorem 3.1 for reflexive H was trivially proved when the diameter of H was ≤ 1, in which case H was a complete graph. Note that for bipartite H, if the diameter of H is ≤ 2 then H is a complete bipartite graph. Also recall that for bipartite H, a graph G is homomorphic to H if and only if G is also bipartite, and H has an edge if G has an edge (if the diameter of H is 0 then H has no edge and contains a single isolated vertex). Similar to the proof for Theorem 3.1, together with the above observation, we see that this theorem holds if the diameter of H is ≤ 2.

Now suppose that the diameter of H is > 2. Let x be any vertex of H such that there exists a vertex at distance > 2 from x in H. There exists such a vertex x in H, as the diameter of H is > 2. The construction of H' is exactly like in the proof of Theorem 3.1 except that now we choose the vertex x as defined above, and there is no loop present in H'. Thus H' is bipartite. Recall that in the construction of H', we are assuming that the maximum distance from x to a vertex of H is s. Thus $s > 2$ here.

In analogy to the reflexive case, we prove the statements (a) and (b) in Theorem 4.1.1 and Theorem 4.1.2 respectively for the case when the diameter of H is > 2.

THEOREM 4.1.1. *RET-H polynomially transforms to COMP-H′.*

Proof. As shown earlier, the theorem is trivially proved when the diameter of H is ≤ 2. We assume here that the diameter of H is > 2. Let a graph G containing H as an induced subgraph be an instance of *RET-H*. Since H is bipartite, only bipartite graphs may be homomorphic to H. Thus for nonbipartite instances of *RET-H*, the theorem is trivially true. Suppose that G is bipartite. We construct in time polynomial in the size of G, a graph G' (containing G and H' as induced subgraphs) such that the following statements (i) and (ii) are equivalent:

(i) G retracts to H.

(ii) G' compacts to H'.

This would imply that *RET-H* polynomially transforms to *COMP-H′*.

Let (G_A, G_B) be a bipartition of G, and (H_A, H_B) be a bipartition of H, with $H_A \subseteq G_A$ and $H_B \subseteq G_B$. Without loss of generality, suppose that $x \in H_A$. The construction of G' is as follows. We add to the subgraph H of G, the vertices of $V(H') - V(H)$ and the edges of $E(H') - E(H)$ such that H is expanded to the graph

H' (the vertices of $H' - H$ are not adjacent to any vertex of $G - H$). Depending on whether s is odd or even, we add paths and edges as described below.

We first assume that s is odd. For each $a \in G_A - H_A$, we add to G a path $U_a = u_1^a u_2^a \dots u_{s-2}^a$ containing $s-2$ new vertices, and we add the edges xu_1^a and au_{s-2}^a (thus we have the path $xU_a a$ of even length $s-1$, and hence $d_{G'}(x,a) \leq s-1$). For each $b \in G_B - H_B$, we add to G a path $U_b = u_1^b u_2^b \dots u_{s-1}^b$ containing $s-1$ new vertices, and we add the edges xu_1^b and bu_{s-1}^b (thus we have the path $xU_b b$ of odd length s, and hence $d_{G'}(x,b) \leq s$).

Now, we assume that s is even. For each $a \in G_A - H_A$, we add to G a path $U_a = u_1^a u_2^a \dots u_{s-1}^a$ containing $s-1$ new vertices, and we add the edges xu_1^a and au_{s-1}^a (thus we have the path $xU_a a$ of even length s, and hence $d_{G'}(x,a) \leq s$). For each $b \in G_B - H_B$, we add to G a path $U_b = u_1^b u_2^b \dots u_{s-2}^b$ containing $s-2$ new vertices, and we add the edges xu_1^b and bu_{s-2}^b (thus we have the path $xU_b b$ of odd length $s-1$, and hence $d_{G'}(x,b) \leq s-1$).

This completes the construction of G'. We thus have $d_{G'}(x,v) \leq s$, for all $v \in V(G-H)$. Now we return to our claim for the equivalence of statements (i) and (ii).

First suppose that $r : G \to H$ is a retraction. We define below a retraction $r' : G' \to H'$, which by definition is also a compaction.

We define

$r'(v) = r(v)$, for all $v \in V(G)$,

$r'(h') = h'$, for all $h' \in V(H') - V(H)$.

We now fix a vertex $v \in V(G-H)$ for defining r' for the vertices of U_v. Let $P_{xv} = h_0 h_1 \dots h_j$ be a shortest path from $r(x)$ to $r(v)$ in H, with $r(x) = h_0$ and $r(v) = h_j$, $h_i \in V(H)$, for all $i = 0, 1, 2, \dots, j$. Note that if $v \in G_A - H_A$ then j is even, as $x, v \in G_A$, and further, $j \leq s-1$ when s is odd, and $j \leq s$ when s is even, as $d_{G'}(x,a) \leq s-1$ when s is odd, and $d_{G'}(x,a) \leq s$ when s is even, for all $a \in G_A - H_A$, and we define $p = s-2$ when s is odd, and $p = s-1$ when s is even. If $v \in G_B - H_B$ then j is odd, as $x \in G_A$, $v \in G_B$, and further, $j \leq s$ when s is odd, and $j \leq s-1$ when s is even, as $d_{G'}(x,b) \leq s$ when s is odd, and $d_{G'}(x,b) \leq s-1$ when s is even, for all $b \in G_B - H_B$, and we define $p = s-1$ when s is odd, and $p = s-2$ when s is even.

For the vertices of U_v, we define

$r'(u_i^v) = h_i$, for all $i = 1, 2, \dots, j-1$,

$r'(u_i^v) = h_j$, and $r'(u_{i+1}^v) = h_{j-1}$, for all $i = j, j+1, \dots, p-1$.

It is easily verified that $r' : G' \to H'$ is a retraction, and hence a compaction.

Conversely, if G' compacts to H' then the proof that G retracts to H is exactly as in the proof of Theorem 3.1.1 for the reflexive case, except when we define c' for the vertices in S, which we now define as follows:

$c'(v) = h$, if $d_{G'}(z,v)$ is odd, $v \in S$,

$c'(v) = h'$, $h' \in V(H')$ is any neighbor of h, if $d_{G'}(z,v)$ is even, $v \in S$. □

THEOREM 4.1.2. *COMP-H′ polynomially transforms to RET-H.*

Proof. As shown earlier, the theorem holds when the diameter of H is ≤ 2. We assume here that the diameter of H is > 2. Let a graph G be an instance of *COMP-H′*. Since H' is bipartite, only bipartite graphs may be homomorphic to H'. Thus for nonbipartite instances of *COMP-H′*, the theorem is trivially true. Suppose that G is bipartite. We construct in time polynomial in the size of G, polynomial (in the size of G) number of graphs $G_1, G_2, \dots, G_\beta$, each containing a copy of H' as an induced subgraph, and for each G_i, we construct in time polynomial (in the size of G_i and hence in the size of G), a graph G_i' also containing a copy of H' as an induced

subgraph, $1 \leq i \leq \beta$, such that the following statements (i), (ii), (iii), and (iv) are equivalent for some value of i, $1 \leq i \leq \beta$:

(i) G compacts to H'.
(ii) G_i retracts to H'.
(iii) G'_i retracts to H'.
(iv) G'_i retracts to H.

We prove that (i) is equivalent to (ii), (ii) is equivalent to (iii), and (iii) is equivalent to (iv). Thus, in effect, we prove that (i) is equivalent to (iv), which shows that *COMP-H′* polynomially transforms to *RET-H* under Turing reduction.

The graphs G_i and G'_i, for all $i = 1, 2, \ldots, \beta$, are constructed exactly the same way as in the proof of Theorem 3.1.2 for the reflexive case. The proof for the equivalence of the statements (i), (ii), (iii), and (iv) is exactly as in the proof of Theorem 3.1.2, except as noted below for defining the retraction $r' : G_i \rightarrow H'$ and corresponding changes for verifying that it is indeed a retraction.

The lists $L_i(v) \subseteq V(H')$, for all $v \in V(G_i)$, are obtained as in the proof of Theorem 3.1.2, as a result of performing the consistency test for G_i with respect to H' and the lists $LST_i(h) = \{h\}$, for all $h \in V(H')$, and $LST_i(v) = V(H')$, for all $v \in V(G_i) - V(H')$, where H' is the copy in G_i, for all $i = 1, 2, \ldots, \beta$.

Suppose that G_i retracts to H', for some value of i, $1 \leq i \leq \beta$. This implies that the consistency test we performed for G_i has succeeded, i.e., $L_i(v) \neq \phi$, for all $v \in V(G_i)$. Let $r : G_i \rightarrow H'$ be a retraction (clearly, $r : G_i \rightarrow H'$ is a list-L_i-homomorphism). We now define another retraction $r' : G_i \rightarrow H'$ as follows. We partition the vertex set of G_i into two sets V_1 and V_2 just as for the reflexive case. Recall that

$V_1 = \{v \in V(G_i) | L_i(v) \subseteq \{h_v\} \cup V(P_{h_v})$, for some vertex $h_v \in V(H)\}$,
$V_2 = V(G_i) - V_1$.

For $v \in V_1$, we let

$r'(v) = h$, where h is the vertex in $L_i(v)$ closest to h_v, that is, $d_{H'}(h, h_v) < d_{H'}(h', h_v)$, for all $h' \in L_i(v), h' \neq h$.

For $v \in V_2$, we let

$r'(v) = h$, if $r(v) \in \{h\} \cup V(P_h)$, and $d_{H'}(r(v), h)$ is even, with $h \in V(H)$,
$r'(v) = h'$, if $r(v) \in \{h\} \cup V(P_h)$, and $d_{H'}(r(v), h)$ is odd, with $h \in V(H)$, and $h' \in V(H)$ is any neighbor of h in H (every vertex in H has a neighbor, as H does not have an isolated vertex).

This completes the definition of r'.

Now we see the changes for verifying that $r' : G_i \rightarrow H'$ is a retraction. We only highlight the differences for the verification as compared to the reflexive case in the proof of Theorem 3.1.2.

While considering the edge ab of G_i, with $a, b \in V_1$, we do not need to consider the case that $d_{H'}(h'_a, h_a) = d_{H'}(h'_b, h_a)$, as this will not happen for bipartite H'. All other arguments for the edge hold as for the reflexive case.

While considering the edge ab of G_i, with $a, b \in V_2$, if $r(a) \in \{h\} \cup V(P_h)$, with $h \in V(H)$, then from the definition of r', we will have $r'(a) = h$ and $r'(b) = h'$, or we will have $r'(a) = h'$ and $r'(b) = h$, where $h' \in V(H)$ is any neighbor of h in H, implying that $r'(a)r'(b)$ is always an edge of H'. All other arguments for the edge hold as for the reflexive case.

While considering the edge ab of G_i, with $a \in V_1$ and $b \in V_2$, if $r(b) \in \{h_a\} \cup V(P_{h_a})$, with $h_a \in V(H)$, then $d_{H'}(r(b), h_a)$ must be odd, as otherwise $h_a \notin L_i(a)$ (due to success of the consistency test), and hence from the definition of r', we will have that $r'(a)r'(b) = h_a h'_a$ is an edge of H', where $h'_a \in V(H)$ is any neighbor of h_a

in H. All other arguments for the edge hold as for the reflexive case. □

We have now proved Theorem 4.1 when H is connected. Suppose now that H may be disconnected. Let $H_1, H_2, \ldots, H_p$, $p \geq 1$, be the components of H. For each component H_i, we construct a graph H_i' analogous to the construction described earlier for the connected case depending on whether the diameter of H_i is ≤ 2 or > 2, $1 \leq i \leq p$. For a graph H_i with diameter > 2, we may assume x_i and s_i to play the role of x and s respectively described earlier for the connected case, $1 \leq i \leq p$. For a graph H_i with diameter ≤ 2, we let x_i be any vertex of H_i, and s_i (≤ 2) is defined similarly, $1 \leq i \leq p$. Our graph H' is the set of components $H_1', H_2', \ldots, H_p'$. We have to prove statements (a) and (b) mentioned at the beginning of the proof of Theorem 4.1.

For proving statement (a), we need to consider the construction of the graph G' in the proof of Theorem 4.1.1. Let $G_1, G_2, \ldots, G_t$ be the components of a graph G, with H_i as an induced subgraph of G_i, for all $i = 1, 2, \ldots, p$. Thus G is an instance of *RET-H*. Since H is bipartite, only bipartite graphs may be homomorphic to H. Thus for nonbipartite instances of *RET-H*, statement (a) is trivially true. Suppose that G is bipartite. Let $(G_{i,A}, G_{i,B})$ be a bipartition of G_i, for all $i = 1, 2, \ldots, t$. Let $(H_{i,A}, H_{i,B})$ be a bipartition of H_i, with $H_{i,A} \subseteq G_{i,A}$, $H_{i,B} \subseteq G_{i,B}$, for all $i = 1, 2, \ldots, p$. Without loss of generality, let $x_i \in H_{i,A}$, for all $i = 1, 2, \ldots, p$.

Suppose first that H has no edge. Then H_i has only one vertex and the diameter of H_i is 0, and hence from our construction, H_i' is the graph H_i, for all $i = 1, 2, \ldots, p$. For G to be homomorphic to H, it must be that G also has no edge, in which case G_i would be the graph H_i, for all $i = 1, 2, \ldots, p$. Clearly, G retracts to H if and only if G compacts to H', and hence statement (a) is true.

Now suppose that H has an edge. Without loss of generality, let H_1 have an edge. If the diameter of H_i is > 2, the construction of G_i' is analogous to the construction described in the proof of Theorem 4.1.1, $1 \leq i \leq p$. If the diameter of H_i is ≤ 2 then G_i' is obtained by adding a path $x_i z_i a$ in G_i, where z_i is a single new vertex added in G_i, for all $a \in G_{i,A} - H_{i,A}$, and by adding an edge $x_i b$ in G_i, for all $b \in G_{i,B} - H_{i,B}$; thus the diameter of G_i' is ≤ 3, $1 \leq i \leq p$. When constructing G_1', we also add a path $x_1 z_1 a$ and an edge $x_1 b$ in G_1, for all $a \in G_{p+1,A}, G_{p+2,A}, \ldots, G_{t,A}$, $b \in G_{p+1,B}, G_{p+2,B}, \ldots, G_{t,B}$, where the vertex z_1 already exists if the diameter of H_1 is ≤ 2, or is a new vertex if the diameter of H_1 is > 2. Notice that the diameter of G_1' is still ≤ 3 if the diameter of H_1 is ≤ 2. Our graph G' is the set of components $G_1', G_2', \ldots, G_p'$. Also note that the diameter of H_i' is ≥ 4 if the diameter of H_i is > 2, and recall that the graph H_i' is the graph H_i if the diameter of H_i is ≤ 2, $1 \leq i \leq p$. We now argue analogous to the arguments in the proof of Theorem 3.1 for the disconnected case for proving statement (a). We however make some remarks as noted below.

Note that if $t > p$ then G_1' has edges as a result of its construction due to the graphs $G_{p+1}, G_{p+2}, \ldots, G_t$. If G retracts to H then in the case when $t > p$, the fact that H_1 has an edge establishes that G_1' is homomorphic to H_1', which is a requirement for G_1' to compact to H_1'.

If G' compacts to H' then in the case when the diameter of H_k is 0, i.e., when H_k has no edge, we point out the reasoning as to why G_k retracts to H_k, $1 \leq i \leq p$. Suppose that the diameter of H_k is 0 for some k, $1 \leq k \leq p$. From our construction, the graph H_k' is the graph H_k. If a component G_i' of G' compacts to H_k' then the graph H_i' must be of diameter 0, as G_i', which contains H_i' as an induced subgraph, must have no edge, $1 \leq i \leq p$. We have that G_k' contains H_k' of diameter 0 as an induced subgraph. Thus if G_k' has an edge then there exists some component H_m' of

diameter 0 to which no component of G' compacts, $1 \leq m \leq p$. Hence if G' compacts to H' then the graph G'_k (and G_k) is the graph H'_k, i.e., H_k, implying that G'_k trivially compacts to H'_k, and G_k trivially retracts to H_k.

For proving statement (b), we just need to assume in the proof of Theorem 4.1.2 that if the diameter of H_i is ≤ 2 then $V(P_h) = \phi$, for all $h \in V(H_i)$, $1 \leq i \leq p$. □

4.1. Compaction to bipartite graphs in relation to constraint satisfaction. As we mentioned in section 3.1, it is shown in [Feder and Vardi, 1998] that for every fixed template T of a set of relations, there exists a fixed bipartite graph H such that the constraint satisfaction problem *CSP-T* and the retraction problem *RET-H* are polynomially equivalent. Since it is thought to be likely difficult to determine whether for every template T, the problem *CSP-T* is polynomial time solvable or NP-complete, this is taken as evidence that to determine whether for every bipartite graph H, the problem *RET-H* is polynomial time solvable or NP-complete is likely to be difficult also.

We have shown in Theorem 4.1 that for every bipartite graph H, there exists a bipartite graph H' such that *RET-H* is polynomially equivalent to *COMP-H'*. Thus from Theorem 4.1 and the above mentioned result of [Feder and Vardi, 1998], we have the following theorem.

THEOREM 4.2. *For every fixed template T of a set of relations, there exists a fixed bipartite graph H such that the constraint satisfaction problem CSP-T and the compaction problem COMP-H are polynomially equivalent.* □

Thus we have evidence, due to Theorem 4.2, that to determine whether for every bipartite graph H, the problem *COMP-H* is polynomial time solvable or NP-complete is likely to be difficult.

REFERENCES

H. J. BANDELT, A. DAHLMANN, AND H. SCHUTTE (1987), *Absolute retracts of bipartite graphs*, Discrete Appl. Math., 16, pp. 191–215.

H. J. BANDELT, M. FARBER, AND P. HELL (1993), *Absolute reflexive retracts and absolute bipartite retracts*, Discrete Appl. Math., 44, pp. 9–20.

T. FEDER AND P. HELL (1998), *List homomorphisms to reflexive graphs*, J. Combin. Theory Ser. B, 72, pp. 236–250.

T. FEDER, P. HELL, AND J. HUANG (1999), *List homomorphisms and circular arc graphs*, Combinatorica, 19, pp. 487–505.

T. FEDER AND M. Y. VARDI (1993), *Monotone monadic SNP and constraint satisfaction*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC), San Diego, CA.

T. FEDER AND M. Y. VARDI (1998), *The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory*, SIAM J. Comput., 28, pp. 57–104.

T. FEDER AND P. WINKLER (1988), manuscript.

W. GUTJAHR, E. WELZL, AND G. WOEGINGER (1992), *Polynomial graph-colorings*, Discrete Appl. Math., 35, pp. 29–45.

F. HARARY (1969), *Graph Theory*, Addison-Wesley, Reading, MA.

P. HELL (1972), *Retractions de Graphes*, Ph.D. thesis, Universite de Montreal, Montreal.

P. HELL (1974), *Retracts in graphs*, in Graphs and Combinatorics, Lecture Notes in Math. 406, Springer-Verlag, Berlin, pp. 291–301.

P. HELL AND D. J. MILLER (1979), *Graphs with forbidden homomorphic images*, Ann. New York Acad. Sci., 319, pp. 270–280.

P. HELL AND J. NESETRIL (1990), *On the complexity of H-colouring*, J. Combin. Theory Ser. B, 48, pp. 92–110.

P. HELL, J. NESETRIL, AND X. ZHU (1996), *Duality and polynomial testing of tree homomorphisms*, Trans. Amer. Math. Soc., 348, pp. 1281–1297.

P. HELL AND I. RIVAL (1987), *Absolute retracts and varieties of reflexive graphs*, Canad. J. Math., 39, pp. 544–567.

A. Karabeg and D. Karabeg (1991), *Graph compaction*, Graph Theory Notes of New York XXI, The New York Academy of Sciences, New York, pp. 44–51.
A. Karabeg and D. Karabeg (1993), *Graph Compaction*, manuscript.
A. K. Mackworth (1977), *Consistency in networks of relations*, Artificial Intelligence, 8, pp. 99–118.
A. K. Mackworth and E. C. Freuder (1985), *The complexity of some polynomial network consistency algorithms for constraint satisfaction problems*, Artificial Intelligence, 25, pp. 65–74.
U. Montanari (1974), *Networks of constraints: Fundamental properties and applications to picture processing*, Inform. Sci., 7, pp. 95–132.
R. Nowakowski and I. Rival (1979), *Fixed-edge theorem for graphs with loops*, J. Graph Theory, 3, pp. 339–350.
E. Pesch (1988), *Retracts of Graphs*, Math. Systems Econom. 110, Athenaum, Frankfurt am Main.
E. Pesch and W. Poguntke (1985), *A characterization of absolute retracts of n-chromatic graphs*, Discrete Math., 57, pp. 99–104.
N. Vikas (1999), *Computational complexity of compaction to cycles*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Baltimore, MD.
N. Vikas (2002), *Connected and loosely connected list homomorphisms*, 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Cesky Krumlov, Czech Republic, Lecture Notes in Comput. Sci. 2573, Springer-Verlag, Heidelberg, Germany, pp. 399–412.
N. Vikas (2003), *Computational complexity of compaction to reflexive cycles*, SIAM J. Comput., 32, pp. 253–280.
N. Vikas (2004), *Computational complexity of compaction to irreflexive cycles*, J. Comput. System Sci., 68, pp. 473–496.

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 783–818

STRICT POLYNOMIAL-TIME IN SIMULATION AND EXTRACTION*

BOAZ BARAK† AND YEHUDA LINDELL‡

Abstract. The notion of efficient computation is usually identified in cryptography and complexity with (strict) probabilistic polynomial-time. However, until recently, in order to obtain *constant-round* zero-knowledge proofs and proofs of knowledge, one had to allow simulators and knowledge extractors to run in time that is only polynomial *on the average* (i.e., *expected* polynomial-time). Recently Barak gave the first constant-round zero-knowledge argument with a *strict* (in contrast to expected) polynomial-time simulator. The simulator in his protocol is a *nonblack-box* simulator (i.e., it makes inherent use of the description of the *code* of the verifier).

In this paper, we further address the question of strict polynomial-time in constant-round zero-knowledge proofs and arguments of knowledge. First, we show that there exists a constant-round zero-knowledge *argument of knowledge* with a *strict* polynomial-time *knowledge extractor*. As in the simulator of Barak's zero-knowledge protocol, the extractor for our argument of knowledge is not black-box and makes inherent use of the code of the prover. On the negative side, we show that nonblack-box techniques are *essential* for both strict polynomial-time simulation and extraction. That is, we show that no (nontrivial) constant-round zero-knowledge proof or argument can have a strict polynomial-time *black-box* simulator. Similarly, we show that no (nontrivial) constant-round zero-knowledge proof or argument of knowledge can have a strict polynomial-time *black-box* knowledge extractor.

Key words. zero-knowledge proof systems, proofs of knowledge, expected vs. strict polynomial-time, black-box vs. nonblack-box algorithms

AMS subject classifications. 94A60, 68Q17, 68Q25

DOI. 10.1137/S0097539703427975

1. Introduction.

This paper deals with the issue of *expected* versus *strict* polynomial-time with respect to simulators and extractors for zero-knowledge proofs and arguments and zero-knowledge proofs and arguments of knowledge.[1]

1.1. Expected polynomial-time in zero knowledge.

The principle behind the definition of (computational) zero-knowledge proofs, as introduced by Goldwasser, Micali, and Rackoff [29], is the following:

> *Anything that an* efficient *verifier can learn as a result of interacting with the prover, can be learned* without interaction *by applying an* efficient *procedure (i.e., simulator) to the public input.*

Note that there are two occurrences of the word "efficient" in this sentence. When providing a formal definition of zero knowledge, the issue of what is actually meant by "efficient computation" must be addressed. The standard interpretation in cryptography and complexity is that of probabilistic polynomial-time. However, in the context

*Received by the editors May 5, 2002; accepted for publication (in revised form) February 2, 2004; published electronically May 5, 2004. An extended abstract of this paper appeared in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, Montreal, QC, Canada, 2002.

http://www.siam.org/journals/sicomp/33-4/42797.html

†School of Mathematics, Institute for Advanced Study, 1 Einstein Drive, Princeton, NJ 08540 (boaz@ias.edu).

‡IBM T.J. Watson Research, 19 Skyline Drive, Hawthorne, NY 10532 (lindell@us.ibm.com). Much of this work was carried out while this author was at the Weizmann Institute of Science, Israel.

[1] In a *proof* system, soundness holds unconditionally and with respect to all-powerful cheating provers. In contrast, in an *argument* system, soundness is only guaranteed to hold with respect to polynomial-time bounded provers. We note that lower bounds for proofs do not necessarily hold for arguments, because in arguments the soundness condition is only computational. Likewise, lower bounds for arguments do not necessarily hold for proofs, because proofs are allowed to have super-polynomial honest prover strategies, whereas arguments are not. See section 2.1.

of zero knowledge, efficiency has also been taken to mean *polynomial on the average* (also known as *expected polynomial-time*). That is, if we fix the input, and look at the running time of the machine in question as a random variable (depending on the machine's coins), then we require only that the *expectation* of this random variable is polynomial. Three versions of the formal definition of zero knowledge appear in the literature, differing in their interpretations of efficient computation:

1. Definition 1—STRICT/STRICT: According to this definition both the verifier and simulator run in strict polynomial-time. This is the definition adopted by Goldreich [22, section 4.3] and is natural in the sense that only the standard interpretation of efficiency is used.
2. Definition 2—STRICT/EXPECTED: This more popular (and liberal) definition requires the verifier to run in strict polynomial-time while allowing the simulator to run in expected polynomial-time. This was actually the definition proposed in the original paper on zero knowledge [29].
3. Definition 3—EXPECTED/EXPECTED: In this definition, *both* the verifier and simulator are allowed to run in expected polynomial-time. This definition is far less standard than the above two, but is nevertheless a natural one to consider. As we describe below, this definition was considered by Feige [17], who showed that (at least for one definition of expected polynomial-time) it is problematic.

As we have mentioned, the standard interpretation of efficient computation is that of (strict) polynomial-time. In light of this, Definition 1 (STRICT/STRICT) seems to be the most natural. Despite this, expected polynomial-time was introduced in the context of zero knowledge because a number of known protocols that could be proven zero knowledge according to the more liberal STRICT/EXPECTED definition were not known to satisfy the more severe STRICT/STRICT definition. In particular, until very recently no *constant-round* zero knowledge argument (or proof) for $\mathcal{NP}$ was known to satisfy Definition 1 (STRICT/STRICT).[2] It was therefore necessary to relax the definition and allow expected polynomial-time simulation (as in Definition 2).

Proofs of knowledge. An analogous situation arises in proofs *of knowledge* [29, 33, 18, 6]. There, the underlying principle is the following:

> *If an* efficient *prover can convince the honest verifier with some probability that $x \in L$, then this prover can apply an* efficient *procedure (i.e., extractor) to x and its private inputs and obtain a witness for x with essentially the same probability.*

Again, the word "efficient" occurs twice, and again three possible definitions can be used. In particular, the prover and extractor can be instantiated by strict polynomial-time machines, expected polynomial-time machines, or a combination of both.

The different definitions—discussion. As has been observed before (e.g., see [17, section 3.2], [22, section 4.12.3]), the definitions that allow for expected polynomial-time computation are not fully satisfactory for several reasons:

- *Philosophical considerations*: Equating "efficient computation" with *expected polynomial-time* is more controversial than equating efficient computation with (strict) probabilistic polynomial-time. For example, Levin [30] (see also [20], [22, section 4.3.1.6]) has shown that when expected polynomial-time is defined as above, the definition is too machine dependent and is not closed under reductions. He proposed a different definition for expected polynomial-time that is closed under reductions and is less machine dependent. However,

[2] We note that throughout this paper we always refer to protocols with negligible soundness error.

it is still unclear whether expected polynomial-time, even under Levin's definition, should be considered as efficient computation.

- *Technical considerations*: Expected polynomial-time is less understood than the more standard strict polynomial-time. This means that rigorous proofs of security of protocols that use zero-knowledge arguments with expected polynomial-time simulators (or arguments of knowledge with expected polynomial-time extractors) as components are typically more complicated (see [31] for an example). Another technical problem that arises is that expected polynomial-time simulation is not closed under composition. Consider, for example, a protocol that uses zero-knowledge as a subprotocol. Furthermore, assume that the security of the larger protocol is proved in two stages. First, the zero-knowledge subprotocol is simulated for the adversary (using an expected polynomial-time simulator). This results in an expected polynomial-time adversary that runs the protocol with the zero-knowledge executions removed. Then, in the next stage, the rest of the protocol is simulated for this adversary. A problem arises because the simulation of the second stage must now be carried out for an expected polynomial-time adversary. However, simulation for an expected polynomial-time adversary can be highly problematic (as the protocol of [24] demonstrates; see [31, Appendix A] for details).
- *Practical considerations*: A proof of security that uses expected polynomial-time simulation does not always achieve the "expected" level of security. For example, assume that a protocol's security relies on a hard problem that would take 100 years to solve, using the best known algorithm. Then, we would like to prove that the probability that an adversary can successfully break the protocol is negligible, unless it runs for 100 years. However, when expected polynomial-time simulation is used, we cannot rule out an adversary that runs for one year and succeeds with probability 1/100. This is a weaker level of security and may not be acceptable. See section 1.4 for a more detailed discussion of this issue.

The liberal STRICT/EXPECTED definition also suffers from a conceptual drawback regarding the notion of zero knowledge itself. Specifically, the idea behind the definition of zero knowledge is that anything that a verifier can learn as a result of the interaction, it can learn by just looking at its input. Therefore, it seems that the simulator should *not* be of a higher complexity class than the verifier. Rather, both the verifier and simulator should be restricted to the same complexity class (i.e., either strict or expected polynomial-time). The EXPECTED/EXPECTED definition has the advantage of not having any discrepancy between the computational power of the verifier and simulator. However, it still suffers from the above described drawbacks with any use of expected polynomial-time. In addition, as Feige [17, section 3.3] pointed out, in order to prove that known protocols remain zero knowledge for expected polynomial-time verifiers, one needs to restrict the verifiers to run in expected polynomial-time not only when interacting with the honest prover but also when interacting with all other interactive machines. This restriction is somewhat controversial because any efficient adversarial strategy should be allowed. In particular, there seems to be no reason to disqualify an adversarial strategy that takes expected polynomial-time when attacking the honest prover, but runs longer otherwise (notice that the adversary is only interested in attacking the honest prover, and so its attack is efficient).

In contrast, the STRICT/STRICT definition suffers from none of the above conceptual difficulties. For this reason, it is arguably a preferred definition. However, as

we have mentioned, it was not known whether this definition can be satisfied by a protocol with a *constant* number of rounds. Thus a natural open question (posed by [17, section 3.4] and [22, section 4.12.3]) was the following:

> *Are expected polynomial-time simulation and extraction necessary in order to obtain constant-round zero-knowledge proofs and proofs of knowledge?*

A first step in answering the above question was recently taken by Barak in [3]. Specifically, [3] presented a zero-knowledge argument system that is both *constant-round* and has a *strict* polynomial-time simulator. Interestingly, the protocol of [3] is not black-box zero knowledge. That is, the simulator utilizes the description of the code of the verifier. (This is in contrast to black-box zero knowledge where the simulator is only given oracle access to the verifier.) Given the existence of non-black-box zero-knowledge arguments with a constant number of rounds and strict polynomial-time simulation, it is natural to ask the following questions:

1. Is it possible to obtain constant-round zero-knowledge *arguments of knowledge* with strict polynomial-time extraction?
2. Is the fact that the protocol of [3] is not black-box zero knowledge coincidental, or is this an inherent property of any constant-round zero-knowledge protocol with strict polynomial-time simulation?

1.2. Our results. In this paper we resolve both the above questions. First, we show that it is possible to obtain *strict* polynomial-time knowledge extraction in a *constant-round* protocol. In fact, we show that it is possible to obtain strict polynomial-time simulation and extraction simultaneously in a zero-knowledge protocol. That is, we prove the following theorem.

THEOREM 1.1. *Assume the existence of collision-resistant hash functions and collections of trapdoor permutations such that the domain of each permutation is the set of all strings of a certain length.*[3] *Then, there exists a* constant-round *zero-knowledge argument of knowledge for $\mathcal{NP}$ with a* strict *polynomial-time knowledge extractor and a* strict *polynomial-time simulator.*

The definition of arguments of knowledge that we refer to in Theorem 1.1 differs from the standard definition of [6] in an important way. In the definition of [6], the knowledge extractor is given only black-box access to the prover. In contrast, in our definition, the knowledge extractor is given the actual description of the prover (i.e., it has nonblack-box access). As we will see below, this modification is actually necessary for obtaining *constant-round* arguments of knowledge with *strict* polynomial-time extraction.

In addition to the above positive result, we show that it is impossible to obtain a (nontrivial) constant-round zero-knowledge protocol that has a strict polynomial-time *black-box* simulator. Likewise, a strict polynomial-time extractor for a constant-round zero-knowledge argument of knowledge cannot be black-box. That is, we prove the following two theorems.

THEOREM 1.2. *There do not exist* constant-round *zero-knowledge proofs or arguments with* strict polynomial-time black-box *simulators for any language $L \notin \mathcal{BPP}$.*

THEOREM 1.3. *There do not exist* constant-round *zero-knowledge proofs or arguments of knowledge with* strict polynomial-time black-box *knowledge extractors for*

[3] By this we mean that there exists a trapdoor permutation family $\{f_s\}_{s\in\{0,1\}^*}$ such that $f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}$. It actually suffices to assume the existence of a family of enhanced trapdoor permutations [23, Appendix C.1]. Such a family can be constructed under the RSA and factoring assumptions; see [2, section 6.2] and [23, Appendix C.1].

any language $L \notin \mathcal{BPP}$.

We therefore conclude that the liberal definitions that allow the simulator (resp., extractor) to run in expected polynomial-time are necessary for achieving constant-round *black-box* zero knowledge (resp., arguments of knowledge). Furthermore, our use of nonblack-box techniques is essential in order to obtain Theorem 1.1.

We note that Theorems 1.2 and 1.3 are tight in the sense that if any super-constant number of rounds is allowed, then zero-knowledge proofs of knowledge with strict polynomial-time black-box extraction and simulation *can* be obtained. This was shown by Goldreich in [22, section 4.7.6]. (Actually, [22] shows that a *super-logarithmic* number of sequential executions of the 3-round zero-knowledge proof for Hamiltonicity [9] suffices. However, using the same ideas, it can be shown that by running $\log n$ parallel executions of the proof of Hamiltonicity, any super-constant number of sequential repetitions is actually enough.)

Zero knowledge versus ε-knowledge. Our impossibility result regarding constant-round black-box zero knowledge with strict polynomial-time simulation has an additional ramification to the question of the relation between black-box ε-knowledge [15] and black-box zero knowledge. Loosely speaking, an interactive proof is called ε-knowledge if for every ε, there exists a simulator that runs in time polynomial in the input and in $1/\varepsilon$, and outputs a distribution that can be distinguished from a real proof transcript with probability at most ε. Despite the fact that this definition seems to be a significant relaxation of zero knowledge, no separation between ε-knowledge and zero knowledge was previously known. Our lower bound indicates a separation for the black-box versions; that is, *black-box* ε-knowledge is strictly weaker than *black-box* zero knowledge. Specifically, on one hand, constant-round black-box ε-knowledge protocols with strict polynomial-time simulators do exist.[4] On the other hand, as we show, analogous protocols for black-box *zero* knowledge do not exist.

Witness-extended emulation. Zero-knowledge proofs of knowledge are often used as subprotocols within larger protocols. Typically, in this context the mere existence of a knowledge extractor does not suffice for proving the security of the larger protocol. Loosely speaking, what is required is the existence of a machine that not only outputs a witness with the required probability (as is required from a knowledge extractor), but also outputs a corresponding simulated transcript of the interaction between the prover and the verifier. Furthermore, whenever the transcript of the interaction is such that the verifier accepts, then the witness that is obtained is valid. To explain this further, consider a case in which the prover convinces the verifier in a real interaction with probability p. Then, with probability negligibly close to p, the aforementioned machine should output an accepting transcript and a valid witness. Furthermore, with probability negligibly close to $1-p$, the machine should output a rejecting transcript (and we don't care about the witness).

This issue was addressed in [31] by Lindell, who called such a machine a "witness-extended emulator." It was proved there that there exists such a witness-extended emulator for any proof of knowledge. However, the extended emulator that is obtained runs in *expected* polynomial-time, even if the original knowledge extractor runs in *strict* polynomial-time. Unfortunately, we do not know how to prove an analogous result that, given any strict polynomial-time knowledge extractor, would provide a

[4] Such a protocol can be constructed by taking any constant-round protocol with an expected polynomial-time simulator and truncating the simulator's run (outputting $\perp$), if it runs for more than $1/\varepsilon$ times its expected running time. By Markov's inequality, the probability of this bad event happening is at most ε.

strict polynomial-time emulator. Instead, we directly construct a strict polynomial-time witness-extended emulator for our zero-knowledge proof of knowledge (under a slightly different definition than [31]).

1.3. On the effect of truncating expected polynomial-time executions. A naive approach to solving the problem of expected polynomial-time in simulation and extraction is to simply truncate the execution of the simulator or extractor after it exceeds its expected running time by "too much." However, this does not necessarily work. The case of knowledge extractors is a good example. Let us fix a proof (or argument) of knowledge for some NP-language L. Let $x \in \{0,1\}^*$, and let P^* be a polynomial-time prover that, for some ϵ, aborts with probability $1-\epsilon$ and convinces the honest verifier that $x \in L$ with probability ϵ. For all previously known constant-round proofs of knowledge, the expected polynomial-time knowledge extractor works in roughly the following way: it first verifies the proof from P^*, and if P^* is not convincing (which occurs in this case with probability $1-\epsilon$), then it aborts. On the other hand, if P^* is convincing (which happens in this case with probability ϵ), then it does expected $p(n) \cdot \frac{1}{\epsilon}$ work (where $p(\cdot)$ is some fixed polynomial) and outputs a witness for x. Clearly, the expected running time of the extractor is polynomial (in particular, it is $p(n)$ plus the time taken to honestly verify a proof). However, if we halt this extractor before it completes $\frac{1}{\epsilon}$ steps, then, with high probability, the extractor will *not* output a witness. Note that $\frac{1}{\epsilon}$ may be much larger than $p(n)$, and therefore the extractor may far exceed its expected running time and yet still not output anything.

In contrast to the above, the knowledge extractor of the argument of knowledge presented in this paper (in section 4) runs in strict polynomial-time which is *independent* of the acceptance probability (i.e., ϵ). For example, if there exists a cheating prover P^* that runs in time n^2, but convinces the verifier that $x \in L$ with probability $\epsilon = n^{-10}$, then our extractor will always run in time, say, n^4 and output a witness with probability at least (negligibly less than) n^{-10}. On the other hand, the extractors for previous protocols would work as follows: with probability $1-n^{-10}$ they would do almost nothing, and with probability n^{-10} they would run for, say, n^{12} steps and output a witness.

1.4. Trading success probability for running time. The observation in section 1.3 about how expected polynomial-time extractors typically work raises serious security issues with respect to the application of proofs of knowledge that have such extractors. For example, suppose that we use a proof of knowledge for an identification protocol based on factoring. Suppose, furthermore, that we use numbers for which the fastest known algorithms will take 100 years to factor. We claim that in this case, if we use a proof of knowledge with an *expected* polynomial-time extractor, then we cannot rule out the possible existence of an adversary that will take one year of computation time and succeed in an impersonation attack with probability 1/100.

In order to see this, notice that the proof of security of the identification protocol works by constructing a factoring algorithm from any impersonator, using the extractor for the proof of knowledge. Thus, for typical protocols, what will actually be proven is that given an algorithm that runs for T steps and successfully impersonates with probability ϵ, we can construct an algorithm that solves the factoring problem with probability ϵ and *expected* running time T. In particular, this factoring algorithm may (and actually will) work in the following way: with probability $1-\epsilon$ it will do nothing and with probability ϵ it will run in T/ϵ steps and factor its input. Thus, the existence of an impersonator that runs for one year and succeeds with probabil-

ity 1/100 only implies the existence of a factoring algorithm that runs for 100 years. Therefore, we cannot rule out such an impersonator. We conclude that the standard proofs of knowledge potentially allow adversaries to trade their success probability for running time. In the concrete example above, the impersonator lowered its running time from 100 years to one year, at the expense of succeeding with probability 1/100 instead of 1. We stress that the fastest known algorithms for factoring *do not* allow such a trade-off. That is, if the parameters are chosen so that 100 years are required to factor, then the probability of successfully factoring after one year is extremely small, and not close to 1/100.

We stress that not only is it the case that the *definition* of expected polynomial-time extraction does not allow us to rule out such an adversary, but also that such adversaries cannot be ruled out by the current proofs of security for known constant-round protocols (thus, the problem also lies with the protocols and not just with the definition). In contrast, such a trade-off is not possible if the extractor runs in *strict* polynomial-time. Rather, an impersonator that runs in time T and succeeds with probability ϵ yields a factoring algorithm that runs in time (polynomially related) to T and succeeds with probability ϵ. Thus, in the above concrete example, an analogous impersonator for a protocol with a strict polynomial-time extractor would yield a factoring algorithm that runs for one year and succeeds with probability 1/100. However, such an algorithm is conjectured not to exist, and therefore such an impersonator also does not exist (unless the conjecture is wrong).

1.5. Further discussion of prior work. Zero-knowledge proofs were introduced by Goldwasser, Micali, and Rackoff [29], and were then shown to exist for all $\mathcal{NP}$ by Goldreich, Micali, and Wigderson [26]. Constant-round zero-knowledge arguments and proofs were constructed by Feige and Shamir [19], Brassard, Crépeau, and Yung [11], and Goldreich and Kahan [24]. All these constant-round protocols utilize expected polynomial-time simulators. Regarding zero-knowledge proofs *of knowledge*, following a discussion in [29], the first formal definitions were provided by Feige, Fiat, and Shamir [18], and by Tompa and Woll [33]. These definitions were later modified by Bellare and Goldreich [6].

The issue of expected polynomial-time is treated in Feige's thesis [17] and Goldreich's book [22]. Goldreich [22, section 4.7.6] also presents a construction for a proof of knowledge with *strict* polynomial-time extraction (and simulation) that uses any super-logarithmic number of rounds (as discussed above, a variant of this construction can be obtained that uses any super-constant number of rounds).

As we have mentioned, until a short time ago, all known constant-round zero-knowledge protocols had expected polynomial-time simulators. However, recently this barrier was broken by Barak [3], who provided the first constant-round zero-knowledge argument for $\mathcal{NP}$ with a *strict* polynomial-time simulator, assuming the existence of collision-resistant hash functions with super-polynomial hardness. Barak and Goldreich [4] later showed how to obtain the same result under the weaker assumption of the existence of standard collision-resistant hash functions (with polynomial-time hardness). The construction of [3] was also the first zero-knowledge argument to utilize a nonblack-box simulator. In a similar fashion, the constant-round argument of knowledge presented in this paper utilizes a nonblack-box *knowledge extractor*. We note that [5] also utilized a nonblack-box knowledge extractor. However, their extractor ran in *expected* polynomial-time, and the nonblack-box access was used there for a completely different reason (specifically, to achieve a *resettable* zero-knowledge argument of knowledge).

1.6. Organization. In section 2 we describe the basic notation and definitions that we use. Then, in section 3 we define and construct a commit-with-extract commitment scheme, which is the main technical tool used to construct our zero-knowledge argument of knowledge. The proof of Theorem 1.1 can be found in section 4 where we present the construction of a zero-knowledge argument of knowledge with strict polynomial-time extraction. Finally, in section 5 we prove Theorems 1.2 and 1.3. That is, we prove that it is impossible to construct strict polynomial-time black-box simulators and extractors for (nontrivial) constant-round protocols.

2. Definitions.

Notation. For a binary relation R, we denote by $R(x)$ the set of all "witnesses" for x. That is, $R(x) \stackrel{\text{def}}{=} \{y \mid (x,y) \in R\}$. Furthermore, we denote by L_R the language induced by the relation R. That is, $L_R \stackrel{\text{def}}{=} \{x \mid R(x) \neq \emptyset\}$.

For a finite set $S \subseteq \{0,1\}^*$, we write $x \in_R S$ to say that x is distributed uniformly over the set S. We denote by U_n the uniform distribution over the set $\{0,1\}^n$.

A function $\mu(\cdot)$ is negligible if for every positive polynomial $p(\cdot)$ and all sufficiently large n's, it holds that $\mu(n) < 1/p(n)$. We let $\mu(\cdot)$ denote an arbitrary negligible function. That is, when we say that $f(n) < \mu(n)$ for some function $f(\cdot)$, we mean that *there exists* a negligible function $\mu(\cdot)$ such that for every n, $f(n) < \mu(n)$. A function $f(\cdot)$ is noticeable if there exists a positive polynomial $p(\cdot)$ such that for all sufficiently large n's, it holds that $f(n) > 1/p(n)$. We note that "noticeable" is not the complement of "negligible."

For two probability ensembles (sequences of random variables) $X = \{X_s\}_{s\in S}$ and $Y = \{Y_s\}_{s\in S}$ (where $S \subseteq \{0,1\}^*$ is a set of strings), we say that X is computationally indistinguishable from Y, denoted $X \stackrel{\text{c}}{\equiv} Y$, if for every polynomial-sized circuit family $\{D_n\}_{n\in\mathsf{N}}$ and every $s \in S$, it holds that $|\Pr[D_{|s|}(X_s) = 1] - \Pr[D_{|s|}(Y_s) = 1]| < \mu(|s|)$. We will sometime drop the subscripts s when they can be inferred from the context. In all our protocols, we will denote the security parameter by n.

Let A be a probabilistic polynomial-time machine. We denote by $A(x,y,r)$ the output of the machine A on input x, auxiliary input y and random tape r. We stress that the running time of A is polynomial in $|x|$.[5] If M is a Turing machine, then we denote by $\mathsf{desc}_n(M)$ the description of a *circuit* that computes M on inputs of size n. Note that a polynomial-time machine that receives $\mathsf{desc}_n(M)$ for input runs in time that is polynomial in the running time of M. Let A and B be interactive machines. We denote by $\mathsf{view}_A(A(x,y,r), B(x,z,r'))$ the view of party A in an interactive execution with machine B, on public input x, where A has auxiliary input y and random tape r, and B has auxiliary input z and random tape r'. The view of party B is denoted similarly. Recall that a party's view of an execution includes the contents of its input, auxiliary input, and random tape plus the transcript of messages that it receives during the execution. We sometimes drop r or r' from this notation, which means that the random tape is not fixed but rather chosen at random. For example, we denote by $\mathsf{view}_A(A(x,y), B(x,z))$ the random variable $\mathsf{view}_A(A(x,y,U_m), B(x,z,U'_{m'}))$, where m (resp., m') is the number of random bits that A (resp., B) uses on input of size $|x|$.

2.1. Zero knowledge. Loosely speaking, an interactive proof system for a language L involves a prover P and a verifier V, where upon common input x, the prover P attempts to convince V that $x \in L$. We note that the prover is often given

[5] We assume that y and r are on different tapes. Therefore, even if y is very long (e.g., $|y| > \text{poly}(|x|)$), it is still possible for A to read r.

some private auxiliary input that "helps" it to prove the statement in question to V. Such a proof system has the following two properties:

1. *Completeness*: this states that when honest P and V interact on common input $x \in L$, then V is convinced of the correctness of the statement that $x \in L$ (except with at most negligible probability).
2. *Soundness*: this states that when V interacts with any (cheating) prover P^* on common input $x \notin L$, then V will be convinced *with at most negligible probability*. (Thus V cannot be tricked into accepting a false statement.)

There are two flavors of soundness: *unconditional* (or statistical) soundness that must hold even for an all-powerful cheating prover, and *computational soundness* that needs hold only for (nonuniform) polynomial-time cheating provers. In *proof* systems [29], unconditional soundness is guaranteed, whereas in *argument* systems [10] only computational soundness must hold. We remark that a proof system is not necessarily an argument system, because the honest prover strategy in a proof system is not required to be polynomial-time (in contrast to arguments where the honest prover as well as the cheating provers must be nonuniform polynomial-time). Unless explicitly stated, when we mention "protocols" in discussion, we mean both proofs and arguments.

Throughout this paper, we will always assume that the soundness error is at most negligible. However, we will not always require this of completeness. Specifically, our lower bounds in section 5 hold even if the completeness error is $1 - 1/p(n)$ for some polynomial $p(\cdot)$; in this case, we will call $p(n)$ the completeness bound.

We now recall the definition of zero knowledge [29]. Actually, we present (a slightly strengthened form of) the definition of *auxiliary-input* zero knowledge [22, section 4.3.3].[6] The main difference between our definition below and the standard definition is that we require the simulator to run in *strict*, rather than *expected*, polynomial-time. We note that in this paper, when we say zero knowledge, our intention is always auxiliary-input zero knowledge.

DEFINITION 2.1 (auxiliary-input zero knowledge). *Let (P, V) be an interactive proof (or argument) system for a language L. Denote by $P_L(x)$ the set of strings y satisfying the completeness condition with respect to $x \in L$ (i.e., when the completeness bound is $p(\cdot)$, then $P_L(x)$ is the set of strings y for which the probability that $\mathsf{view}_V(P(x,y), V(x))$ is accepting is at least $p(|x|)$). We say that (P, V) is* auxiliary-input zero knowledge *if there exists a strict probabilistic polynomial-time algorithm S such that for every strict probabilistic polynomial-time machine V^* it holds that*

$$\left\{\mathsf{view}_{V^*}(P(x,y), V^*(x,z))\right\}_{x\in L, y\in P_L(x), z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{S(\mathsf{desc}_{|x|}(V^*), x, z)\right\}_{x\in L, y\in P_L(x), z\in\{0,1\}^*}$$ [7]

Black-box zero knowledge. A zero-knowledge proof system is called black-box zero knowledge [27] if the simulator S only uses its input $\mathsf{desc}(V^*)$ as a black-box

[6] We deviate from the definition of auxiliary-input zero knowledge of [22, section 4.3.3] by making the slightly stronger requirement that there exists a single universal simulator for all verifiers, rather than a different simulator for each verifier as in [22, section 4.3.3]. Note however, that the definition of [22, section 4.3.3] already implies that for any $c > 0$ there exists a universal simulator for all $\mathbf{Time}(n^c)$ verifiers.

[7] Recall that $\mathsf{desc}_n(M)$ is the description of a circuit that computes M on inputs of size n. An equivalent formulation provides the simulator S with the description of the actual Turing machine M. However, in this case, it is also necessary to provide S with 1^t, where t is a bound on the running time of V^* on inputs of length $|x|$. This additional input is provided in order to allow S to run in time which is (some fixed) polynomial in the running time of V^*.

subroutine. That is, S is an oracle algorithm such that

$$\left\{\mathsf{view}_{V^*}(P(x,y),V^*(x,z))\right\}_{x\in L,y\in P_L(x),z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{S^{V^*(x,z,\cdot,\cdot)}(x)\right\}_{x\in L,y\in P_L(x),z\in\{0,1\}^*},$$

where $V^*(x,z,\cdot,\cdot)$ denotes the *next-message function* of the interactive machine V^* when the common input x and auxiliary input z are fixed (i.e., the next-message function of V^* receives a random tape r and a message history h and outputs $V^*(x,z,r,h)$).

2.2. Zero-knowledge arguments of knowledge. Our definition of proofs and arguments of knowledge below differs from the standard definition of [6] in two ways:

Strict polynomial-time extraction. We require that the knowledge extractor run in *strict* polynomial-time (rather than in expected polynomial-time).

Nonblack-box extraction. The knowledge extractor is given access to the *description* of the prover. This is a relaxation of the standard definition of proofs of knowledge (cf. [6, 22]) in which the knowledge extractor is given *only* oracle (or black-box) access to the prover strategy. The relaxed definition appeared originally in Feige and Shamir [19] (which differs from the definition in [18]; see discussion in [6]), and suffices for all practical applications of arguments of knowledge.

Until recently, all known proofs of knowledge (including [19]) were coupled with an extractor that used the prover algorithm only as a black-box. The extra power of nonblack-box extraction (where the knowledge extractor is given the actual description of the prover) was first used in an essential way by [5] in order to obtain resettable zero-knowledge arguments of knowledge for $\mathcal{NP}$.[8] We show in section 5 that our use of nonblack-box extraction is also essential, as there do not exist constant-round proofs of knowledge with *black-box* strict polynomial-time extractors.

We are now ready to present the following definition.

DEFINITION 2.2 (system of proofs/arguments of knowledge). *Let R be a binary relation. We say that a probabilistic, polynomial-time interactive machine V is a* knowledge verifier for the relation R *with* negligible knowledge error *if the following two conditions hold:*

- Nontriviality: *There exists a probabilistic polynomial-time*[9] *interactive machine P such that for every $(x,y)\in R$, all possible interactions of V with P on common input x, where P has auxiliary input y, are accepting.*
- Validity (or knowledge soundness) with negligible error: *There exists a* strict *probabilistic polynomial-time machine K such that for every strict probabilistic polynomial-time machine P^* and every $x,y,r\in\{0,1\}^*$, machine K satisfies the following condition:*

  Denote by $p(x,y,r)$ the probability (over the random tape of V) that V accepts upon input x, when interacting with the prover P^* that has input x, auxiliary input y, and random tape r. Then, machine K, upon input $(\mathsf{desc}_{|x|}(P^*),x,y,r)$, outputs a solution $s\in R(x)$ with probability at least $p(x,y,r)-\mu(|x|)$.

[8] The use there is critical as it can be shown that if the knowledge extractor is restricted to only black-box access to the prover, then resettable zero-knowledge arguments of knowledge are possible for languages in $\mathcal{BPP}$ only; see [12].

[9] The requirement that P be polynomial-time is inherent for arguments, but not for proofs. A proof system with such a prover is called an efficient-prover proof.

An interactive pair (P, V) such that V is a knowledge verifier for a relation R and P is a machine satisfying the nontriviality condition (with respect to V and R) is called an argument of knowledge *for the relation R. If the validity condition holds with respect to any (not necessarily polynomial-time) machine P^*, then (P, V) is called a* proof of knowledge *for R.*

If an argument (resp., proof) of knowledge (P, V) is zero knowledge for the language L_R induced by R, then we say that (P, V) constitutes a system of zero-knowledge arguments (resp., proofs) of knowledge for R.

2.3. Witness-extended emulation. In this section, we present an extension of the notion of proofs of knowledge, called witness-extended emulation. This extension is of importance when zero-knowledge proofs or arguments of knowledge are used as subprotocols within larger protocols, as is often the case. Typically in this context, the extractor for the proof of knowledge supplies the simulator for the larger protocol with some secret information. This information is then used in the proof of security of the rest of the larger protocol.

The final output of the simulator for the larger protocol is usually a transcript of the entire simulated protocol execution (where this transcript is indistinguishable from a real execution). Thus, the extractor needs to not only extract a witness from the proof of knowledge, but must also obtain a matching transcript of the execution of the proof of knowledge itself. However, by definition, the extractor only outputs a witness and does not provide the simulator with such a transcript. This issue was addressed in [31] where, loosely speaking, it was shown that for any zero-knowledge proof of knowledge, there exists a machine that outputs both the witness (with the appropriate probability) *and* a matching transcript of messages sent in the execution. Such a machine was termed a "witness-extended emulator" because its role is to emulate a protocol execution while also providing a witness (see [31] for a more detailed discussion). We proceed by presenting a slightly different definition of witness-extended emulation and then discuss its relevance to our work. We begin with some notation and terminology:

- Recall that $\mathsf{view}_{P^*}(P^*(x, y, r), V(x))$ denotes a random variable describing the view of P^* in a protocol execution with the honest verifier V, where P^* has input x, auxiliary input y, and random tape r, and the honest verifier V has input x. (This random variable depends only on the coins of V.)
- We say that a zero-knowledge protocol (P, V) is *publicly verifiable* if given the transcript of messages between any P^* and V, it is possible to efficiently determine whether or not V accepted the proof. (For example, any protocol can be made publicly verifiable by having the verifier send the contents of its random tape at the end of the protocol execution. Note, however, that this can affect other properties of the protocol. Indeed, our proof of knowledge, Protocol 4.1, *cannot* be made publicly verifiable in this way while still preserving the witness-extended emulation property.)

  When a protocol is publicly verifiable, then V's accept/reject bit can be deduced efficiently (and deterministically) from the prover's view. We denote by $\mathsf{accept}_V(\cdot)$ the deterministic function that takes a specific view of the prover in a protocol execution, and outputs whether or not V accepts in this execution.

We are now ready to present the following definition.

DEFINITION 2.3 (witness-extended emulator). *Let R be a binary relation, and let (P, V) be an interactive proof system that is* publicly verifiable. *Consider a probabilis-*

tic machine E that is given the description of a probabilistic polynomial-time prover $\mathsf{desc}_{|x|}(P^*)$, and the contents of P^*'s input, auxiliary input, and random tapes, x, y and r, respectively. We denote by $E_1(\mathsf{desc}_{|x|}(P^*), x, y, r)$ and $E_2(\mathsf{desc}_{|x|}(P^*), x, y, r)$ the random variables representing the first and second elements of the output of E, respectively. We say that E is a witness-extended emulator for (P, V) and R *if it runs in* strict *polynomial-time and if for every probabilistic polynomial-time interactive machine P^*, every $y, r \in \{0,1\}^*$, and all sufficiently large x's, the following hold:*

1. *E_1's output distribution is indistinguishable from the distribution of the view of P^* in a real execution with the honest verifier V. That is,*

$$\left\{E_1(\mathsf{desc}_{|x|}(P^*), x, y, r)\right\}_{x,y,r} \stackrel{\mathrm{c}}{\equiv} \left\{\mathsf{view}_{P^*}(P^*(x, y, r), V(x))\right\}_{x,y,r}.$$

2. *The probability that V accepts in the view of P^* that is output by E_1, and yet E_2 does not output a correct witness, is negligible. That is,*

$$\begin{aligned}\Pr\big[&\mathsf{accept}_V(E_1(\mathsf{desc}_{|x|}(P^*), x, y, r)) \\ &= 1 \ \&\ E_2(\mathsf{desc}_{|x|}(P^*), x, y, r) \notin R_L(x)\big] < \mu(|x|).\end{aligned}$$

Definition 2.3 differs from the definition of [31] in a number of ways. Most notably, we provide the witness-extended emulator with the description of P^* (rather than just black-box access) and also require it to run in strict polynomial-time (rather than expected polynomial-time). The other differences are technical and are mainly due to the need to achieve strict polynomial-time emulation. Despite the differences, the definitions are the same in spirit and both achieve the desired goal of enabling a proof of knowledge to be used as a subprotocol in some other, larger protocol.

Recall that [31] proved the existence of a witness-extended emulator for any proof of knowledge. However, the emulator that is obtained runs in expected polynomial-time and thus does not achieve our goal of strict polynomial-time emulation. We will therefore directly prove the existence of a witness-extended emulator for our zero-knowledge argument of knowledge.

3. Commitment with extraction. In order to construct a constant-round zero-knowledge argument of knowledge with strict polynomial-time extraction, we first construct a new primitive that we call commit-with-extract. Loosely speaking, a commit-with-extract scheme is a commitment scheme with the additional property that the committed value can be extracted from the sender. More precisely, there exists a (strict polynomial-time) commitment extractor that is given the description of the sender (along with the contents of its input, auxiliary input, and random tape) and extracts the value being committed to during the commit stage of the protocol. This notion of "extractable commitments" is not completely new. It has been used in the context of secure multiparty computation (e.g., [21, Construction 2.3.8]) and has been called both commit-with-knowledge [14] and nonoblivious commitment [22, Definition 4.9.3]. One main technical difference between our primitive and previous ones is the requirement that the extractor run in *strict* polynomial-time. At the end of this section we discuss another significant difference that is related to the question of "knowledge" versus "extraction."

We remark that in a standard commitment scheme, the committer may not "know" the value that it committed to. This is the case, for example, in the case that the range of the commitment scheme is $\{0,1\}^*$, and so any value is a valid

commitment. Clearly, such a commitment scheme does *not* have the property that the committed value can be extracted from the committer.

3.1. Definition. We begin by informally defining perfectly binding commitment schemes.

Commitment schemes. A commitment scheme is a two-party protocol that enables a party, known as the *sender*, to commit itself to a value while keeping it secret from the *receiver*. A commitment scheme must be both *hiding* and *binding*. The hiding property of a commitment scheme says that the receiver's view in the case that the sender commits to 0 is computationally indistinguishable from its view in the case that the sender commits to 1. (Thus, the committed value is unknown to the receiver.) The binding property of a commitment scheme states that in a later stage when the commitment is opened, the "opening" can yield only a single value that was determined in the committing phase. (Thus, the sender cannot modify its committed value.) In a *perfectly binding* commitment scheme, the binding property states that transcripts resulting from a commitment to 0 and a commitment to 1 must be *disjoint*. Thus, for any given transcript, there is at most one commitment value that can yield that transcript. See [22, section 4.4.1] for a formal definition of commitment schemes.

Commit-with-extract. As we have mentioned, a commit-with-extract scheme is a commitment scheme with the following additional property: there exists a (strict polynomial-time) commitment extractor that is given the description of the sender and extracts the value being committed to during the commit stage of the protocol. In addition to outputting the committed value, we also require the extractor to output the sender's view of an execution (this is similar in spirit to witness-extended emulation and is needed when the commit-with-extract is used as a subprotocol).[10] Of course, the committed value and sender's view output by the extractor must be compatible. In order to enforce this compatibility, we denote by $\mathsf{commit\text{-}value}(\cdot)$ a function that takes a sender's view and outputs the *unique* committed value implicit in this view, or $\perp$ if no such value exists. (This function is well defined for perfectly binding commitments because in such a case every transcript can define at most one value.) Then, compatibility between the view and committed value is obtained by requiring that $x = \mathsf{commit\text{-}value}(v)$, where x and v are the committed value and sender's view, respectively, as output by the extractor. We now present the formal definition.

DEFINITION 3.1 (commit-with-extract). *A* perfectly binding *commitment scheme C (with sender A and receiver B) is a* commit-with-extract *commitment scheme if the following holds: there exists a strict probabilistic polynomial-time commitment extractor CK such that for every probabilistic polynomial-time committing party A^* and for every $x, y, r \in \{0,1\}^*$, upon input $(\mathsf{desc}_{|x|}(A^*), x, y, r)$, machine CK outputs a pair, denoted $(CK_1(\mathsf{desc}_{|x|}(A^*), x, y, r), CK_2(\mathsf{desc}_{|x|}(A^*), x, y, r))$, satisfying the following conditions:*

1. $\{CK_1(\mathsf{desc}_{|x|}(A^*), x, y, r)\}_{x,y,r\in\{0,1\}^*} \stackrel{\mathrm{c}}{\equiv} \{\mathsf{view}_{A^*}(A^*(x, y, r), B)\}_{x,y,r\in\{0,1\}^*}$.
2. $\Pr[CK_2(\mathsf{desc}_{|x|}(A^*), x, y, r) = \mathsf{commit\text{-}value}(CK_1(\mathsf{desc}_{|x|}(A^*), x, y, r))] > 1 - \mu(|x|)$.

We note that the requirements on the extractor CK can be relaxed such that in the case that there is no committed value (i.e., where the view v is such that $\mathsf{commit\text{-}value}(v) = \perp$), then CK can output any arbitrary value, and not just $\perp$. This relaxation suffices for our applications.

[10] The extractor outputs the view of the *sender* (and not the receiver) because the extraction procedure is used in the simulation of a corrupted sender (not receiver).

Commit-with-extract using proofs of knowledge. We note that it is possible to achieve a commit-with-extract scheme in the following straightforward way. First, the sender sends a standard perfectly binding commitment to the receiver. Then, the sender proves knowledge of the committed value using a zero-knowledge proof or argument of knowledge. A commitment extractor can easily be constructed for this scheme by having it run the knowledge extractor from the proof of knowledge and obtain the committed value. However, as mentioned above, known constructions of proofs of knowledge with strict polynomial-time extraction have a nonconstant number of rounds. In contrast, our aim is to construct a commit-with-extract scheme that has a *constant* number of rounds.

Public decommitment. We say that a commitment scheme satisfies public decommitment if the validity of a decommitment can be ascertained by any party that holds the transcript of the messages between the sender and receiver from the commitment stage. In particular, this party need not know the random coins used by the receiver during the commitment. More formally, the specification of a commitment scheme consists of two sender protocols and two receiver protocols: one protocol for the commit phase and one for the reveal phase. In general, the input of a protocol in the reveal phase may be the *entire view* of the corresponding party from the protocol of the commit phase. We say that a commitment scheme satisfies public decommitment if the input of the *receiver* protocol in the reveal phase may consist merely of the *transcript of messages* sent by the parties in the commit phase, and nothing else. That is, see the following definition.

DEFINITION 3.2 (commitment schemes with public decommitment). *A commitment scheme satisfies* public decommitment *if the receiver's decision in the reveal phase depends only on the transcript of messages sent between the parties.*

This notion is analogous to that of "publicly verifiable" protocols, as described in section 2.3. (Again, as described in section 2.3, any perfectly binding commitment scheme can be modified into one that provides public decommitment by having the receiver send its random coins at the conclusion of the commitment phase. However, this may affect other properties of the commitment scheme. Indeed, the extractor for our commit-with-extract scheme would fail to generate a view that is indistinguishable from the sender's view, as required in Definition 3.1, if the receiver were required to send all of its random coins when the commitment phase concludes.) We use the additional feature of public decommitment for constructing a zero-knowledge argument of knowledge from a commit-with-extract scheme.

3.2. Constant-round commit-with-extract. In this section we show how to construct a constant-round commit-with-extract commitment scheme. That is, we prove the following theorem.

THEOREM 3.3. *Assume the existence of collision-resistant hash functions and collections of trapdoor permutations such that the domain of each permutation is the set of all strings of a certain length.*[11] *Then, there exist constant-round commit-with-extract string commitment schemes satisfying public decommitment.*

Before presenting our scheme, we note that it suffices to present a scheme that is perfectly binding, *except with negligible probability* (where the probability is taken over the coins of the receiver). A perfectly binding scheme (with no error) can then be obtained by augmenting the commitment phase with an additional perfectly binding commitment. Specifically, the sender will commit to the same value twice, once using

[11] See footnote 3.

a perfectly binding scheme (that does not enable extraction but is perfectly binding with no error), and once using a commit-with-extract scheme (that enables extraction but has a negligible error with respect to binding). The decommitment phase is then also modified so that both commitments must be opened. The result is a commit-with-extract scheme that is perfectly binding with no error. (We note that perfect binding with negligible error usually suffices, and so no such augmentation is really necessary.)

We now present our construction. In order to simplify the presentation, we start by showing a commit-with-extract *bit* commitment scheme and then show how to generalize our construction to a *string* commitment scheme. Our protocol is based on the following well-known noninteractive commitment scheme that uses one-way permutations [8]: Let f be a one-way permutation over $\{0,1\}^n$ and let b be a hard-core predicate of f. Then, in order to commit to a bit σ, the sender chooses $r \in_R \{0,1\}^n$, lets $y = f(r)$, and sends $\langle y, b(r) \oplus \sigma \rangle$ to the receiver. Loosely speaking, our commitment scheme is similar except that the value y is chosen *jointly* by the sender and the receiver using a coin-tossing protocol (which is based on the coin-tossing protocol of [31]). Since y is uniformly distributed, the hiding property remains as in the original scheme. Likewise, because f is a permutation, y defines a unique value $b(f^{-1}(y))$ and thus the scheme remains perfectly binding. The novelty of our scheme is that for every sender, there exists an extractor that can bias the coin-tossing protocol such that it concludes with a value y for which the extractor knows the preimage $r = f^{-1}(y)$. In this case, the extractor can easily obtain the commitment value σ, as desired.

In order to allow the sender to be implementable by an efficient algorithm, we choose f to be a *trapdoor* one-way permutation. Thus, the sender is able to efficiently compute $r = f^{-1}(y)$, where y is the output of the coin-tossing protocol (this is similar to the noninteractive zero-knowledge protocol constructed in [16]). Formally, the protocol is parameterized by a family of trapdoor permutations over $\{0,1\}^n$, with a function sampling algorithm I. We denote a permutation from the family by f and its associated trapdoor by t. Furthermore, we denote by b a hard-core of f.

One of the components of the protocol is a constant-round zero-knowledge argument with a *strict* polynomial-time simulator. We note that such an argument exists if collision-resistant hash functions exist [3, 4]. As we have mentioned above, another component of our commit-with-extract protocol is a coin-tossing subprotocol that is based on the protocol of [31]. We do not plug in the exact protocol of [31] (while replacing the zero-knowledge proofs with those that run in strict polynomial-time), because its proof of security uses *extraction* from a proof of knowledge, and no proof of knowledge with strict polynomial-time extraction was previously known (indeed, providing such a proof is the aim of our construction).

PROTOCOL 3.4 (commit-with-extract bit commitment scheme).

- **Input:** *The sender has a bit σ to be committed to.*
- **Commit phase:**
  1. A chooses a trapdoor permutation:
     (a) *The sender A chooses a trapdoor permutation f along with its trapdoor t (by running the sampling algorithm I on a uniformly chosen string $s_I \in_R \{0,1\}^n$), and sends f to the receiver B.*
     (b) *A proves to B that f is a permutation, using any constant-round zero-knowledge proof or argument (even one with an expected polynomial-time simulator). For example, if I is such that it outputs a permutation with probability 1, then A may prove that there exists*

a string s_I such that f is the permutation output from $I(s_I)$.[12] *If B does not accept the proof, then it aborts.*

2. A and B run a coin-tossing protocol:
   (a) *B chooses a random string $r_1 \in_R \{0,1\}^n$ and sends $c = \mathsf{Commit}(r_1; s)$ to A (where $\mathsf{Commit}(\cdot)$ denotes any perfectly binding commitment scheme, and $\mathsf{Commit}(r_1; s)$ denotes a commitment to value r_1 using random coins s).*
   (b) *A chooses a random string $r_2 \in_R \{0,1\}^n$ and sends r_2 to B.*
   (c) *B sends r_1 to A (without decommitting).*
   (d) *B proves that the string r_1 sent in Step 2(c) is indeed the value that it committed to in Step 2(a), using a constant-round zero-knowledge argument with a* strict *polynomial-time simulator. Formally, B proves that there exists a string s such that $c = \mathsf{Commit}(r_1; s)$.*
   (e) *The output of the coin-tossing phase is $r_1 \oplus r_2$.*
3. A sends the actual commitment:
   A computes $r = f^{-1}(r_1 \oplus r_2)$ and sends B the value $v = b(r) \oplus \sigma$.

- **Reveal phase:**
  1. *A sends B the string r.*
  2. *B checks that $f(r) = r_1 \oplus r_2$. If this is the case, then B computes $b(r) \oplus v$ obtaining σ. Otherwise, B outputs $\perp$.*

(By convention, if the commit phase of the protocol is not completed, then the committed value is defined to equal 0.) We now prove that Protocol 3.4 is a secure commit-with-extract commitment scheme. We first show that it is a secure *commitment scheme*. This involves demonstrating both the binding and hiding properties. Intuitively, these properties hold because the only difference between the above protocol and the basic commitment scheme defined by $C_n(\sigma; r) \stackrel{\text{def}}{=} \langle f(r), b(r) \oplus \sigma \rangle$ for $r \in_R \{0,1\}^n$ is that the random string r is chosen via a coin-tossing protocol (rather than being determined by the sender).

PROPOSITION 3.5. *Protocol* 3.4 *is a secure bit commitment scheme with public decommitment.*

Proof. We first claim that Protocol 3.4 satisfies public decommitment (see Definition 3.2). This is due to the fact that the only information needed by B to verify A's decommitment is the pair of strings r_1 and r_2 that appear in the transcript between A and B from the commit phase.

Next, we prove the (almost) perfect binding property. That is, we show that except with negligible probability, for any transcript of messages trans generated by an execution between an arbitrary probabilistic polynomial-time sender A^* and the honest receiver B, there exists a *unique* value $\sigma \in \{0,1\}$ such that $\mathsf{commit\text{-}value}(\mathsf{trans}) = \sigma$. Intuitively, the perfect binding property holds as long as the function f sent by A^* is a permutation. This is the case because when f is a permutation, the values r_1 and r_2 in the transcript define a unique value $r = f^{-1}(r_1 \oplus r_2)$, which in turn uniquely defines the value $\sigma = v \oplus b(r)$. The formal argument follows.

First, note that if B does not accept the proof provided by A^* in Step 1(b), then B will abort and then, by convention, σ equals 0. Likewise, if A^* does not complete

[12] We note that by using the primality testers of [28] or [1], for example, the sampling algorithms for both the RSA and Rabin families of trapdoor permutations can be made to output a permutation with probability 1. In the case that the sampling algorithm does not output a permutation with probability 1, a different method of proving that f is a permutation is needed. General methods for achieving this can be found in [7].

the entire commit phase, σ also equals 0. Therefore, the binding property trivially holds in these cases. We continue to show that it holds when B does not abort and the commit phase is completed.

Now, assume that the function f sent by A^* is a permutation. In this case, any pair of strings r_1 and r_2 appearing in the transcript define a single preimage $r = f^{-1}(r_1 \oplus r_2)$. Therefore, any bit v sent by A^* in Step 3 defines a single value $\sigma = v \oplus b(r)$. That is, the values r_1, r_2, and v in the transcript define a single committed value σ, as required. This, however, holds only as long as f is a permutation (otherwise, there may be more than one possible preimage to $r_1 \oplus r_2$). By the soundness of the proof (or argument) of Step 1(b), we have that the probability that f is not a permutation is at most negligible. We therefore conclude that, except with negligible probability, the transcript defines a single committed value.

We now turn to the computational hiding property. Intuitively, the hiding property follows from the hiding property of the noninteractive commitment scheme of [8] and the security of the coin-tossing protocol. In particular, if $r_1 \oplus r_2$ is uniform (or pseudorandom), then distinguishing between a commitment to 0 and a commitment to 1 is essentially equivalent to distinguishing between $\{f(U_n), b(U_n)\}$ and $\{f(U_n), b(U_n) \oplus 1\}$. Since b is a hard-core of f, it is infeasible to distinguish between these distributions in polynomial time. The hiding property therefore follows from the security of the coin-tossing protocol that ensures that $r_1 \oplus r_2$ is pseudorandom.

In the above intuition, the security of the coin-tossing protocol is reduced to a single instance of a commitment. However, in the actual proof, we reduce the indistinguishability of our commitment scheme to the indistinguishability of *multiple samples* of the basic commitment scheme relative to a single permutation f. That is, we show that distinguishing between a commitment to 0 and a commitment to 1 is equivalent to distinguishing between the random variables X_0^m and X_1^m, where $X_\sigma^m = \langle f, f(U_n^{(1)}), b(U_n^{(1)}) \oplus \sigma, \ldots, f(U_n^{(m)}), b(U_n^{(m)}) \oplus \sigma \rangle$, where f is a trapdoor one-way permutation, b is a hard-core bit for f, and $m = \text{poly}(n)$. One can use a standard hybrid argument to show that X_0^m and X_1^m are computationally indistinguishable.

Formally, for any polynomial-size receiver B^*, denote by $v_n^{B^*}(\sigma)$ the distribution over B^*'s view, when the honest sender A commits to the value σ (the distribution is over the uniform choice of random coins for A). Then, the hiding property is stated as follows: for any polynomial-size receiver B^*, it holds that

$$\left\{v_n^{B^*}(0)\right\}_{n\in\mathsf{N}} \stackrel{\text{c}}{\equiv} \left\{v_n^{B^*}(1)\right\}_{n\in\mathsf{N}}.$$

Assume by contradiction that there exists a polynomial-size receiver B^*, a polynomial-time distinguisher D, and a polynomial $p(\cdot)$ such that for infinitely many n's

$$\mathsf{adv}_n^D \stackrel{\text{def}}{=} \left|\Pr[D(v_n^{B^*}(0)) = 1] - \Pr[D(v_n^{B^*}(1)) = 1]\right| \geq \frac{1}{p(n)}. \tag{3.1}$$

We will use D and B^* to construct a distinguisher D' that will distinguish between the random variables X_0^m and X_1^m mentioned above, for $m = 5p(n)$. To get an intuition for the operation of D', consider the case in which B^* doesn't behave in an ostensibly faulty way (i.e., B^* does not "abort" the computation). If this is the case, we can actually construct a distinguisher D' for the basic commitment scheme (i.e., a distinguisher between X_0^1 and X_1^1). The distinguisher D' receives a commitment $C_n(\sigma) = \langle f(r), b(r) \oplus \sigma \rangle$ for input (where $r \in_R \{0,1\}^n$) and works by invoking B^* and running an execution of Protocol 3.4 with B^*, until B^* sends a commitment

$c = \mathsf{Commit}(r_1; s)$ as part of the coin-tossing protocol. At this point, D' learns r_1 by running the continuation of the coin-tossing protocol with B^*. After learning r_1, algorithm D' rewinds B^* back to the point after B^* sent the commitment to r_1, and then D' feeds B^* with the message $r_2 = f(r) \oplus r_1$, where $f(r)$ is the first part of its input commitment $C_n(\sigma) = \langle f(r), b(r) \oplus \sigma \rangle$. The result of the coin-tossing protocol is thus $r_1 \oplus r_2 = f(r)$, which means that as the final message D' can send B^* its input $b(r) \oplus \sigma$. We see that if D distinguishes between the result of this experiment when $\sigma = 0$ and when $\sigma = 1$, then D' breaks the basic commitment scheme C_n and distinguishes between X_0^1 and X_1^1. Unfortunately, the fact that B^* may ostensibly misbehave makes the formal proof somewhat more complicated than this description. We are now ready for the formal description of the distinguisher D'.

Distinguisher D' receives for input a trapdoor one-way permutation f and a sequence

$$\langle f, f(r^{(1)}), b(r^{(1)}) \oplus \sigma, \ldots, f(r^{(m)}), b(r^{(m)}) \oplus \sigma \rangle,$$

where $r^{(1)}, \ldots, r^{(m)}$ are independently and randomly distributed in $\{0,1\}^n$. Algorithm D' then simulates an execution of A with B^* as follows:

1. *Simulation of A choosing a trapdoor permutation*:
   (a) D' passes B^* the permutation f that it was given as part of its input.
   (b) D' runs the zero-knowledge simulator (for the proof that f is a permutation) using the residual B^* as the verifier.[13]
2. *Sample an execution of the coin-tossing protocol*:
   (a) D' receives a commitment c from B^* (c is supposed to equal $\mathsf{Commit}(r_1)$ for some r_1).
   (b) D' chooses a random string $r_2 \in_R \{0,1\}^n$ and passes it to B^*.
   (c) D' obtains some string r_1 from B^*.
   (d) D' verifies the zero knowledge argument given by B^*. If D' accepts the argument from B^*, then it continues. Otherwise, D' sets Z to be the partial transcript until the point that the execution aborted and jumps to Step 5.
3. *Iterate until successful simulation*: D' does the following for $i = 1, 2, \ldots, m$:
   (a) D' rewinds B^* to the point after B^* sent the commitment c (i.e., Step 2(a)) and sends B^* the string $r_2 = f(r^{(i)}) \oplus r_1$, where r_1 is the value obtained in Step 2(c), and $f(r^{(i)})$ comes from D''s input sequence.
   (b) D' obtains some string r_1' from B^* and verifies the zero knowledge argument given by B^*. There are three possibilities at this point:
      i. If D' does not accept the argument from B^*, then it lets $i \leftarrow i + 1$ and continues on to the next iteration (i.e., it returns to Step 3(a)). If the maximum number of attempts have elapsed (i.e., if $i = m$), then D' halts and outputs fail.
      ii. If D' does accept the argument from B^* but $r_1' \neq r_1$, then D' halts and outputs fail.
      iii. If D' accepts the argument from B^* and $r_1' = r_1$, then D' proceeds to Step 4 below.

[13] Note that *any* constant-round zero-knowledge proof or argument may be used for this step in the protocol. Therefore, the simulation of this argument by D' may require expected (rather than strict) polynomial-time. We therefore obtain a distinguisher that runs in expected polynomial-time. This suffices here because in the context of distinguishing commitments (or solving a hard problem), it is possible to truncate D''s execution without lowering its success below what is needed for deriving a contradiction.

4. *Simulation of the actual commitment*: D' passes the bit $v^i = b(U_n^{(i)}) \oplus \sigma$ (from its input sequence) to B^*. Algorithm D' lets Z denote the transcript of the simulated execution and proceeds to Step 5.
5. *Output*: D' passes the transcript Z of the simulated execution to D and outputs whatever D does.
   (We stress that this transcript (as passed by D' to D) may not be complete, as in the case that D' does not accept the argument from B^* in Step 2(d).)

To prove that D' distinguishes between X_0^m and X_1^m with nonnegligible probability, it suffices to prove the following claim:

CLAIM 3.5.1. *Let Z_σ be the random variable that denotes the simulated transcript that D' feeds to D in Step* 5*, when D gets X_σ^m as input. Let Y_σ be the random variable that denotes the view of B^* in an interaction with the sender A when the sender commits to σ. Then Z_σ and Y_σ are $\frac{2}{m}$-computationally indistinguishable. That is, for every polynomial-sized circuit C,*

$$\Big|\Pr[C(Z_\sigma) = 1] - \Pr[C(Y_\sigma) = 1]\Big| < \frac{2}{m}.$$

Claim 3.5.1 is sufficient to show that D' distinguishes between X_0^m and X_1^m because our contradiction hypothesis is that (for infinitely many n's) the distinguisher D distinguishes between Y_0 and Y_1 with gap $\frac{1}{p(n)} = \frac{5}{m}$. Thus, Claim 3.5.1 implies that D will also distinguish between Z_0 and Z_1 with gap at least $\frac{1}{m}$, and hence D' will distinguish between X_0^m and X_1^m with this gap (in contradiction to their computational indistinguishability). We now prove the claim.

To prove Claim 3.5.1, we will use a hybrid argument with two intermediate random variables $\tilde{Z}_\sigma$ and $\hat{Z}_\sigma$ that are computationally indistinguishable from Z_σ and Y_σ, respectively. We then show that $\tilde{Z}_\sigma$ and $\hat{Z}_\sigma$ are $1/m$-indistinguishable.

The random variable $\tilde{Z}_\sigma$. The random variable $\tilde{Z}_\sigma$ is defined to be the result of the following process: Let $\tilde{D}$ be an algorithm that behaves exactly like D' except for the following two differences. First, instead of using a *simulated* zero-knowledge proof in Step 1(b), it uses the honest prover algorithm (we assume that $\tilde{D}$ is given the trapdoor information for the one-way permutation). Second, if $\tilde{D}$ accepts the argument from B^* but $r_1' \neq r_1$, as in Step 3(b)ii, then it does *not* output fail. Rather, it proceeds to Step 4 and uses its knowledge of the trapdoor in order to complete the commitment like an honest committer (note that $\tilde{D}$ can derive the value of σ from its input sequence because it knows the trapdoor). We define $\tilde{Z}_\sigma$ to be the corresponding value computed by $\tilde{D}$ on input X_σ^m in Step 5.

Assuming that $\tilde{D}$ does not accept the argument from B^* when $r_1' \neq r_1$ (i.e., the case in Step 3(b)ii does not happen), the random variable $\tilde{Z}_\sigma$ is computationally indistinguishable from Z_σ because the only difference is whether the zero-knowledge protocol of Step 1(b) is simulated or real. It therefore suffices to show that the probability that $\tilde{D}$ accepts an argument from B^* when $r_1' \neq r_1$ is at most negligible (i.e., the case in Step 3(b)ii happens with at most negligible probability). In order to see this, notice that the case in Step 3(b)ii happens if $r_1 \neq r_1'$ and, in addition, $\tilde{D}$ accepted an argument from B^* that $c = \mathsf{Commit}(r_1)$ (in Step 2(d)) and also an argument from B^* that $c = \mathsf{Commit}(r_1')$ (in Step 3(b)). However, the commitment c that $\tilde{D}$ receives from B^* in Step 2(a) is perfectly binding. Therefore, c defines a single decommitment value; in particular, it cannot be a commitment to both r_1 and r_1'. This means that one of the statements $c = \mathsf{Commit}(r_1)$ and $c = \mathsf{Commit}(r_1')$ is *false*. Therefore, by the soundness of the zero-knowledge argument, $\tilde{D}$ will accept both

arguments from B^* (in Steps 2(d) and 3(b)) with at most negligible probability. We conclude that for large enough n, no polynomial-sized circuit can distinguish between $\tilde{Z}_\sigma$ and Z_σ with nonnegligible probability.

The random variable $\hat{Z}_\sigma$. The random variable $\hat{Z}_\sigma$ is defined by considering the output of the algorithm $\hat{D}$. This algorithm behaves like $\tilde{D}$ except that it does not halt after m iterations. Rather, it continues until it accepts the argument from B^* in Step 3(b) (irrespective of whether or not $r_1 = r_1'$). According to the above description, in the ith iteration, algorithm $\tilde{D}$ sends $r_2 = f(r^{(i)}) \oplus r_1$ to B^*. Thus, for the first $i \leq m$ iterations $\hat{D}$ works in the same way as $\tilde{D}$ and also sends $r_2 = f(r^{(i)}) \oplus r_1$ to B^*. However, if $\hat{D}$ exceeds m iterations, it cannot compute r_2 in this way (because its input includes only m commitments of the form $\langle f(r^{(i)}), b(r^{(i)}) \oplus \sigma \rangle$). Rather, it chooses r_2 uniformly at random (like the honest committer) and sends it to B^*. Then, if it accepts the argument from B^*, algorithm $\hat{D}$ proceeds to Step 4, computes $f^{-1}(r_1 \oplus r_2)$, and commits to σ like the honest committer. (Recall that, like $\tilde{D}$, machine $\hat{D}$ knows the trapdoor and so it can compute f^{-1} and can also obtain the value σ from its input sequence of commitments.) We stress that $\hat{D}$ uses fresh random choices for r_2 in every iteration $i > m$.

We claim that the statistical distance between $\hat{Z}_\sigma$ and $\tilde{Z}_\sigma$ is at most $\frac{1}{m}$. Indeed, one can see that the expected number of iterations that algorithm $\hat{D}$ runs is at most 1: let π be the partial execution of the protocol obtained by $\hat{D}$ until the point that B^* sends the commitment to r_1, and let p_π be the probability that B^* successfully proves the argument to $\hat{D}$, when continuing from the partial execution π. Then, the probability that $\hat{D}$ enters Step 3 equals p_π (because if B^* does not convince $\hat{D}$ in Step 2(d), then $\hat{D}$ does not enter Step 3). Now, within the iterations of Step 3, the probability that B^* convinces $\hat{D}$ is exactly p_π. Therefore, given that $\hat{D}$ enters Step 3, the expected number of iterations is $\frac{1}{p_\pi}$. We conclude that the overall expected number of iterations made by $\hat{D}$ in Step 3 conditioned on the initial partial execution being π equals $p_\pi \cdot \frac{1}{p_\pi} = 1$. Averaging over all partial executions we obtain that the overall expected number of repetitions is also 1, and thus by Markov's inequality, we have that the probability that $\hat{D}$ will require more than m iterations is less than $\frac{1}{m}$. Notice finally that if $\hat{D}$ does not require more than m iterations, then it generates exactly the same distribution as $\tilde{D}$. Therefore, the statistical difference between $\tilde{Z}_\sigma$ and $\hat{Z}_\sigma$ is at most $\frac{1}{m}$.

Comparing $\hat{Z}_\sigma$ with Y_σ. Finally, we claim that the random variable $\hat{Z}_\sigma$ is identical to the variable Y_σ, which denotes the transcript of a real execution. In order to see this, observe that $\hat{D}$ essentially does the following. It first samples a partial execution until the point that B^* sends its commitment to r_1 (this sample is identical to a real execution). Next, it samples the remainder of the execution. If B^* does not convince $\hat{D}$ in the argument that it provides in this sample, then $\hat{D}$ outputs the transcripts and halts. In contrast, if B^* does convince $\hat{D}$ in this sample, then $\hat{D}$ continues until it obtains another sample in which B^* is convincing (all of this sampling is identical to a real execution). Assuming that $\hat{D}$ eventually halts, we have that the result is identical to Y_σ. The fact that $\hat{D}$ eventually halts follows from the fact that it enters Step 3 only if there is a nonzero probability of B^* convincing $\hat{D}$.

Combining the above, we have that the random variables Z_σ and Y_σ can be distinguished with advantage at most $\frac{1}{m} + \mu(n) < \frac{2}{m}$. With this the proof of Claim 3.5.1, and hence the proof of the computational hiding property, is completed. □

Having proven that Protocol 3.4 is a secure commitment scheme, we proceed to show that it is also a *commit-with-extract* scheme, by demonstrating the existence of

a *strict* polynomial-time extractor.

PROPOSITION 3.6. *Protocol* 3.4 *constitutes a commit-with-extract commitment scheme.*

Proof. Intuitively, the extractor CK works by biasing the outcome $r_1 \oplus r_2$ of the coin-tossing protocol such that it knows the preimage under f. More specifically, CK chooses a random string r, computes $f(r)$, and then makes the output $r_1 \oplus r_2$ equal $f(r)$. This is clearly not possible for a real receiver (as the coin-tossing protocol ensures that $f(r)$ is pseudorandom). However, recall that CK has the description of the sender A^*, and therefore has more power than a real receiver. In particular, this gives CK the capability of running the simulator for the proof that B provides in Step 2(d) of the protocol. As we will see, this is enough.

Recall that CK should output a view indistinguishable from the one seen by A^* in a real interaction, as well as the unique commitment value defined by this view. Extractor CK receives the description of an arbitrary polynomial-time sender A^* and a triple (x, y, r), and works as follows:

1. *A^* chooses a trapdoor permutation*:
   (a) CK invokes $A^*(x, y, r)$ and receives the description of a permutation f from A^*.
   (b) Next, CK verifies the zero-knowledge proof from A^* attesting to the fact that f is a permutation. If the verification fails, then CK outputs A^*'s view until this point along with the value 0, and halts. (CK outputs 0 because, by our convention, in an aborted execution this is the default committed value.)
2. *CK biases the outcome of the coin-tossing protocol*:
   (a) CK passes $c = \mathsf{Commit}(0^n)$ to A^* (this is a commitment to "garbage").
   (b) CK obtains a string r_2 from A^*.
   (c) CK chooses $r \in_R \{0,1\}^n$, computes $f(r)$, and passes A^* the string $r_1 = f(r) \oplus r_2$. (Notice that r_1 is distributed uniformly and independently of the initial commitment c, and that $f^{-1}(r_1 \oplus r_2) = r$.)
   (d) CK invokes the zero-knowledge simulator, with the residual A^* as the verifier, for the appropriate (false) statement that there exists a string s such that $c = \mathsf{Commit}(r_1; s)$.
3. *A^* sends the actual commitment*:
   CK receives a bit v from A^*.
4. *Output*: CK outputs A^*'s view of the above execution along with $\sigma = b(r) \oplus v$.

We first claim that CK extracts the bit committed to in the execution (we focus on the case that A^* does not abort; otherwise the claim trivially holds). This is immediate because CK knows the preimage under f of $f(r_1 \oplus r_2) = f(r)$. Therefore, $b(r) \oplus v$ is exactly the unique value committed to by A^*. (The above assumes that f is indeed a permutation. However, by the soundness of the zero-knowledge argument of Step 1(b), it can only occur that f is not a permutation with negligible probability.)

Next, we show that the view output by CK is computationally indistinguishable from A^*'s view in a real execution. These two distributions differ in two aspects: first, the commitment received by A^* in Step 2(a) is to 0^n rather than to r_1. Second, the zero-knowledge argument verified by A^* in Step 2(d) is simulated rather than real. Using a standard hybrid argument, computational indistinguishability can be shown. Specifically, define a hybrid experiment whereby A^* receives a commitment to r_1 (instead of to 0^n), and yet the zero-knowledge proof that it verifies is simulated. Then, by the hiding property of commitment schemes, A^*'s view in the hybrid experiment is indistinguishable from its view output by CK. On the other hand, by the

indistinguishability of zero-knowledge simulation, A^*'s view in the hybrid experiment is indistinguishable from its view in a real execution. We conclude that CK outputs a view that is indistinguishable from A^*'s view in a real execution.

It remains to show that CK runs in strict polynomial-time. However, this immediately follows from the above description and from the fact that the zero-knowledge simulator used by CK runs in strict polynomial-time. This completes the proof. □

We note that any constant-round zero-knowledge argument system with a strict polynomial-time simulator suffices for Step 2(d) of Protocol 3.4. However, the only known such argument system is that of [3] (or its modified version in [4]) and this system utilizes a *nonblack-box* simulator. Since the extractor must run this simulator, it follows that it is also nonblack-box. As we will see in section 5, this is in fact necessary for obtaining a constant-round protocol with strict polynomial-time extraction. (Recall that any constant-round zero-knowledge protocol suffices for Step 1(b), even one with expected polynomial-time simulation.)

Extending Protocol 3.4 *to strings.* To prove Theorem 3.3 we need to generalize Protocol 3.4 to allow commitments to strings of length m (where m is polynomial in the security parameter n), instead of just allowing commitments to single bits. This extension can be obtained in two ways. First, one can simply run Protocol 3.4 in parallel m times (taking care that the zero-knowledge arguments used are closed under the parallel composition of m executions, as is the case with the bounded-concurrent zero-knowledge protocol of [3, 4]). Alternatively, one can directly modify Protocol 3.4 and have A and B run m copies of the coin-tossing protocol in parallel, and then use only a single zero-knowledge argument to prove a compound statement relating to all copies.

Discussion—knowledge versus extraction. As we have mentioned, the notion of a commitment scheme with the additional property that the committed value can be extracted from the sender is not new. However, all previous constructions worked by first committing to a value and then proving knowledge of this committed value. In contrast, Protocol 3.4 works in a completely different way: it does not consist of two distinct "commit" and "knowledge/extract" phases; rather the commitment and extraction are intertwined. Interestingly, this results in a subtle, yet important difference.

In order to exemplify this, consider the following sender A^*. Sender A^* obtains a one-way permutation and erases the corresponding trapdoor. Then, in Step 3 of Protocol 3.4, A^* sends a *random* bit $b \in \{0,1\}$ to B. Otherwise, A^* follows the instructions for A (and successfully concludes the commit stage). The important observation here is that since $r_1 \oplus r_2$ is uniformly distributed, A^* cannot guess the value of the bit that it itself committed to with probability nonnegligibly greater than 1/2. This is very strange, especially considering the fact that extraction is supposed to imply "knowledge."

Such a phenomenon cannot occur in an extractable commitment scheme that uses separate commit and extract phases. This is because the committed value is determined *before* the extraction begins. Therefore, the sender can apply the extractor to itself and thereby obtain the committed value. Thus, it makes sense to say that the sender "knows" the committed value. However, in our protocol, the committed value is determined only at the *conclusion* of the protocol. Furthermore, the value that is committed to may also depend on the random coins of the receiver B (as is indeed the case for the above-described sender A^*). Therefore, it is not possible for the sender to later "apply the extractor to itself in order to obtain the committed value."

Thus, on one hand, it seems that the intuition that the sender "knows" the

committed value cannot be justified here. On the other hand, it seems that for applications that use such schemes, it suffices to extract the actual committed value, irrespective of whether or not the value is predetermined.

Open problem. Our construction of a commit-with-extract scheme requires trapdoor permutations (for finding the preimage to $r_1 \oplus r_2$), and collision-resistant hash functions (for the constant-round zero-knowledge argument with strict polynomial-time simulation [3, 4]). In contrast, when the commitment extractor may run in *expected* polynomial-time or when a *nonconstant* number of rounds may be tolerated, such a scheme can be constructed based on one-way functions only. This raises an interesting question as to whether a constant-round commit-with-extract scheme can be constructed from one-way functions only.

4. A zero-knowledge argument of knowledge. Given a constant-round commit-with-extract commitment scheme, it is not hard to construct a constant-round zero-knowledge argument of knowledge with strict polynomial-time extraction, thereby proving Theorem 1.1. The basic idea is that the prover commits to a witness using the commit-with-extract scheme, and then proves that it has committed to a valid witness using a zero-knowledge proof of membership. Intuitively, soundness follows from the soundness of the zero-knowledge proof of membership and from the fact that the commit-with-extract scheme is perfectly binding. Zero knowledge follows from the hiding property of the commit-with-extract scheme and from the zero-knowledge property of the proof of membership. Lastly, a knowledge extractor is immediately obtained from the extractor of the commit-with-extract scheme. Details follow.

Let R be an NP-relation. Without loss of generality, we assume that all witnesses for R are of the same length. We construct a zero-knowledge argument of knowledge for R as follows.

PROTOCOL 4.1 (zero-knowledge argument of knowledge for R).

- Common input: x.
- Auxiliary input to prover: *w such that $(x, w) \in R$.*
- Phase 1: *P and V run a commit-with-extract protocol (with public decommitment) in which P commits to the witness w.*
- Phase 2: *P proves to V, using a* constant-round *zero-knowledge proof (or argument) of membership (with a strict polynomial-time simulator) that it has committed to a valid witness w in the previous step.*
  Formally, let trans *be the transcript of the commit-with-extract execution of Phase* 1*, and let d be the decommitment message that P would send to V in the decommit phase of the commit-with-extract scheme. Then, P proves the NP-statement that there exists a value d such that* (trans, d) *defines a value w, such that $(x, w) \in R$.*[14]

In Phase 2, we use a zero-knowledge argument system with a strict polynomial-time simulator so that the resulting protocol will be a zero-knowledge argument of knowledge with both a strict polynomial-time extractor and a strict polynomial-time simulator. If the system used in Phase 2 has an *expected* polynomial-time simulator, then the resulting protocol will have a strict polynomial-time knowledge extractor but an *expected* polynomial-time simulator.

Theorem 1.1 is obtained by combining the following proposition with Theorem 3.3 (i.e., the existence of commit-with-extract schemes).

[14]Here we use the assumption that the commit-with-extract scheme satisfies public decommitment as in Definition 3.2. Otherwise, the statement that P needs to prove is not guaranteed to be in $\mathcal{NP}$.

PROPOSITION 4.2. *Assume that the commitment scheme of Phase* 1 *is a* constant-round *commit-with-extract string commitment scheme. Then, Protocol* 4.1 *is a* constant-round *zero-knowledge argument of knowledge for* R*, as defined in Definitions* 2.1 *and* 2.2 *(i.e., it has a strict polynomial-time simulator and a strict polynomial-time knowledge extractor). Furthermore, there exists a witness-extended emulator for Protocol* 4.1*, as defined in Definition* 2.3.

Proof. In order to prove that Protocol 4.1 is a zero-knowledge argument of knowledge, we prove three properties: completeness, knowledge soundness (which implies computational soundness), and zero knowledge. The proof of completeness is immediate. We proceed to prove zero knowledge and knowledge soundness.

Zero knowledge. The simulator S that we build to demonstrate the zero-knowledge property works as follows. In Phase 1 of the protocol, S follows the honest sender strategy of the commit-with-extract scheme, but instead of committing to a real witness w (which it does not have), it commits to garbage (e.g., to all zeros). Next, in Phase 2, simulator S cannot prove that it committed to a correct witness (because it indeed did not). Rather, S runs the simulator for the zero-knowledge proof of Phase 2. The hiding property of the commit-with-extract scheme implies that the commitment to garbage is indistinguishable from a commitment to a real witness. In addition, the zero-knowledge property of the proof of membership implies that the simulation is indistinguishable from a real proof of membership. Combining these together (and using a standard hybrid argument), we obtain that the overall simulation by S is indistinguishable from a real execution of Protocol 4.1. Formally, let V^* be a verifier algorithm for Protocol 4.1. Then the simulator S works as follows.

ALGORITHM 4.3 (simulator S).

- Input: $x \in L$, $z \in \{0,1\}^*$.

1. *S plays the honest sender for the commit-with-extract scheme with $V^*(x,z)$ as the receiver, and commits to $0^{p(|x|)}$ (where $p(n)$ is the length of all witnesses for statements of length n).*
2. *Let* trans *be the series of messages sent to V^* in the previous step, and denote by $V^*(x,z,\mathsf{trans})$ the residual machine that verifies the proof in Phase* 2.[15] *Then, S runs the simulator for the zero-knowledge proof of Phase* 2*, with $V^*(x,z,\mathsf{trans})$ as the verifier.*
3. *Output whatever V^* outputs (without loss of generality, we assume that V^* always outputs its view).*

We need to prove that

$$\{S(x,z)\}_{x\in L, z\in\{0,1\}^*} \stackrel{\mathrm{c}}{\equiv} \{\mathsf{view}_{V^*}(P(x,y),V^*(x,z))\}_{x\in L, y\in R_L(x), z\in\{0,1\}^*},$$

where P is the honest prover algorithm. This is proven using a standard hybrid argument. We define an intermediate distribution $H_{x,y,z}$ in the following way: $H_{x,y,z}$ is produced by an algorithm S' that follows the honest prover's strategy in the first phase and the simulator's strategy in the second phase (in order to enable it to run the honest prover's strategy in the first phase, it is explicitly given a valid witness y). That is, on input (x,y,z), where $(x,y) \in R$, algorithm S' runs the commit-with-extract algorithm and commits to the value y as the honest prover does (instead of to $0^{p(|x|)}$ as the simulator S would). However, in Phase 2, algorithm S' runs the zero-

[15] The residual machine $V^*(x,z,\mathsf{trans})$ is formally defined as the machine that upon receiving a sequence of messages $(\alpha_1,\ldots,\alpha_i)$ replies with the message that $V^*(x,z)$ would send upon receiving the sequence of messages $(\mathsf{trans},\alpha_1,\ldots,\alpha_i)$.

knowledge simulator on $V^*(x, z, \mathsf{trans})$ exactly as S does (instead of really proving the statement as the honest prover would).

The fact that $\{S(x,z)\} \stackrel{c}{\equiv} \{H_{x,y,z}\}$ follows directly from the hiding (secrecy) property of the commit-with-extract scheme. The fact that $\{H_{x,y,z}\} \stackrel{c}{\equiv} \{\mathsf{view}_{V^*}(P(x,y), V^*(x,z))\}$ follows directly from the (auxiliary-input) zero-knowledge property of the proof of membership used in Phase 2. These two facts together imply that $\{S(x,z)\} \stackrel{c}{\equiv} \{\mathsf{view}_{V^*}(P(x,y), V^*(x,z))\}$, as required.

We note that the simulator S inherits the properties of the underlying simulator for the proof of Phase 2. Thus, if the underlying simulator is strict polynomial-time and nonblack-box, then so too is S. On the other hand, if the underlying simulator is black-box or runs in expected polynomial-time, then the same is true for S.

Knowledge soundness. Let P^* be a (possibly cheating) prover that convinces the honest verifier that $x \in L$ with probability ϵ. The extractor for the zero-knowledge argument simply uses the extractor CK of the commit-with-extract scheme in order to obtain P^*'s view of the first phase, along with a string w that is the unique value that is committed to in this phase. Intuitively, w must be a valid witness with probability at least $\epsilon - \mu(|x|)$, for some negligible function $\mu(\cdot)$. This holds because in Phase 2 of the protocol, P^* proves the validity of the committed witness. It then follows from the soundness of the proof of membership that if w is not a valid witness, the verifier rejects (except with negligible probability). In the formal proof of this, we also use the fact that P^* "behaves" in a similar way in its interaction with CK and in a real interaction with the honest verifier. (If this was not the case, then P^* may always convince V, but never commit to a valid witness with CK. The extractor CK would then never extract a valid witness from P^*.) We note that this fact regarding the similarity of P^*'s behavior with CK and V follows from the fact that CK also outputs a view that is computationally indistinguishable to P^*'s view in a real interaction with V. We now formally describe the knowledge extractor algorithm K.

ALGORITHM 4.4 (knowledge extractor K).

- Input: $(\mathsf{desc}_{|x|}(P^*), x, y, r)$.

1. *Let CK be the extractor for the commit-with-extract scheme. Then, invoke CK on input $(\mathsf{desc}_{|x|}(P^*), x, y, r)$, and obtain the view of P^* in Phase 1, denoted v, along with a string w that is the committed value corresponding to that view (i.e., $w = \mathsf{commit\text{-}value}(v)$).*
2. *Output w.*

Let $p(x,y,r)$ be the probability that $P^*(x,y,r)$ convinces the honest verifier on input x in a real execution. We claim that the probability that the witness output by K is valid is at least $p(x,y,r) - \mu(|x|)$, for some negligible function $\mu(\cdot)$. In order to show this, consider a modified extractor $\tilde{K}$ that has the same Step 1 as K, but in Step 2 it first verifies the proof of membership provided by P^* (in Phase 2 of the protocol) and then outputs w if and only if it accepts this proof. Formally, in Step 1, $\tilde{K}$ runs the extractor CK and obtains a pair (v, w), as described for K. Then, in Step 2, $\tilde{K}$ verifies the proof of membership given in Phase 2 by the residual prover P^* that is defined by the view v (i.e., the residual prover is the original P^*, but with its current view set to v). Extractor $\tilde{K}$ then outputs w if and only if it accepts this proof.

Now, since $\tilde{K}$ only contains additional checks, it follows that K outputs w whenever $\tilde{K}$ outputs w. Therefore, it suffices to show that $\tilde{K}$ outputs a correct witness w with probability at least $p(x,y,r) - \mu(|x|)$. In order to demonstrate that $\tilde{K}$ succeeds with this probability, first recall that by the definition of a commit-with-extract

scheme, it holds that

$$\left\{\mathsf{view}_{P^*}^{CK}(x,y,r)\right\} \stackrel{c}{\equiv} \left\{\mathsf{view}_{P^*}(P^*(x,y,r),V)\right\},$$

where $\mathsf{view}_{P^*}^{CK}(x,y,r)$ is the random variable describing the view of P^* as output by CK, and $\mathsf{view}_{P^*}(P^*(x,y,r),V)$ is the random variable describing the view of P^* in a real execution with V. Therefore, the probability that $\tilde{K}$ accepts the proof of membership from the residual P^* is negligibly close to the probability that the honest verifier V accepts this proof of membership. (Otherwise, it is possible to distinguish between P^*'s real view and the view output by CK by emulating an execution of the proof between the residual prover P^* and the verifier $\tilde{K}$, and then outputting 1 if and only if $\tilde{K}$ accepts. This emulation is carried out by running the residual prover that is defined by the view v and seeing if the honest verifier accepts. Note that this emulation can be carried out because P^* is polynomial-time and, in addition, the description of P^* and the view v fully define the residual prover. Therefore, the residual prover can be obtained and run in polynomial-time.) Thus, $\tilde{K}$ accepts the proof of membership with probability at least $p(x,y,r) - \mu(|x|)$. By the soundness of this proof of membership, it must be that w is a valid witness with probability at least $p(x,y,r) - \mu'(|x|)$, for some negligible function μ' (otherwise, the residual prover is able to prove a false statement with nonnegligible probability). This completes the proof of knowledge soundness.

Witness-extended emulation. We conclude by providing a proof of the existence of a witness-extended emulator E for Protocol 4.1. Recall that, upon input $(\mathsf{desc}_{|x|}(P^*), x, y, r)$, E must output a view that is indistinguishable from $P^*(x,y,r)$'s view in a real execution. Furthermore, if this view contains a transcript in which the honest verifier V accepts the proof, then E must also output a valid witness (except with negligible probability). This is easily accomplished as follows:

Emulator E extracts a witness in the same way as the extractor K described above. However, E must also output P^*'s complete view of an execution. In order to do this, after CK concludes, E proceeds to verify the proof of membership of Phase 2 (like the aforementioned $\tilde{K}$). This suffices to obtain P^*'s entire view: the commit-with-extract extractor CK provides P^*'s view from Phase 1, and the remainder of P^*'s view is derived from the verification of the proof of membership of Phase 2.

More formally, the witness-extended emulator E invokes the commit-with-extract extractor CK with input $(\mathsf{desc}_{|x|}(P^*), x, y, r)$ and obtains the output. This output contains a view v that is indistinguishable from P^*'s view in a real execution of commit-with-extract. Furthermore, if this view defines a committed value, then CK also outputs this value (with overwhelming probability). Next, E verifies the proof of membership from the residual prover P^* that is defined by the view v (as described above for $\tilde{K}$). In this proof, E obtains the residual P^*'s view of Phase 2. Then, E concatenates the view output by CK in Phase 1 with P^*'s view in Phase 2, and outputs them both as P^*'s view of the entire execution. Furthermore, if E accepted the proof of Phase 2, then it outputs the witness obtained from CK in Phase 1.

It is easy to see that the view output by E is indistinguishable from P^*'s view in a real execution (P^*'s view from Phase 1 is indistinguishable from its view in a real execution, and the view from Phase 2 is identical). Furthermore, if E accepts the proof of Phase 2, then with overwhelming probability the transcript of Phase 1 defines a valid witness. As we have mentioned above, by the properties of CK, it follows that with overwhelming probability it also outputs this witness. This concludes the proof. □

5. Black-box lower bounds. In this section, we show that there does not exist a constant-round zero-knowledge argument (resp., argument of knowledge), with a black-box simulator (resp., extractor) that runs in strict polynomial-time. That is, we prove Theorems 1.2 and 1.3.

Before presenting the proofs, we provide intuition as to why it is not possible to obtain a strict polynomial-time black-box extractor for constant-round zero-knowledge protocols. First, consider a very simple (cheating) prover P^* that at every step either aborts or sends the honest prover message. Furthermore, the probability that it does not abort at any given step is $\epsilon = \epsilon(n)$. Then, black-box extractors for constant-round zero-knowledge protocols typically extract a witness using the following strategy: Invoke an execution with P^* and if P^* aborts, then also abort. However, if P^* does not abort (and thus sends a prover message in some crucial round), then continually rewind P^* until another prover message is obtained for this round. It is essential that the extractor obtains at least two different prover messages, because this is what gives it additional power over an honest verifier. (Additional power is essential because an honest verifier should not learn anything from the prover, whereas the extractor must obtain the actual witness.) Now, for P^* described above, the extractor enters the "rewinding stage" with probability only ϵ. However, once it enters this stage, the expected number of rewinding attempts by the extractor until a second nonabort reply is obtained equals $1/\epsilon$ (because P^*'s nonabort probability at every step is only ϵ). We therefore have that the overall expected amount of work is bounded. However, we cannot provide *any* strict polynomial upper bound on the running time of the simulator, because for every given polynomial, it is possible to choose ϵ so that $1/\epsilon$ is greater than this polynomial.

This idea underlies our lower bounds. Specifically, we show that by carefully choosing the abort probabilities, it is possible to achieve the following effect: A strict polynomial-time black-box extractor will not have "time" to obtain two nonabort responses from the prover P^* in any given round. (By "not having time," we mean that with noticeable probability, the extractor will have to wait longer than the bound on its running time in order to see a second nonabort response.) Essentially, this means that the extractor cannot "rewind" the prover. However, as we have mentioned above, an extractor must have additional power over a real verifier, and the only additional power awarded a *black-box* extractor is essentially the ability to rewind the prover. We therefore conclude that strict polynomial-time black-box extractors cannot exist for constant-round zero-knowledge protocols. The same argument also holds for strict polynomial-time simulation of constant-round zero-knowledge protocols. We now proceed to the proofs.

THEOREM 5.1 (Theorem 1.2—restated). *There do not exist* constant-round *zero-knowledge proofs or arguments with* strict polynomial-time black-box *simulators for any language* $L \notin \mathcal{BPP}$.

THEOREM 5.2 (Theorem 1.3—restated). *There do not exist* constant-round *zero-knowledge proofs or arguments of knowledge with* strict polynomial-time black-box *knowledge extractors for any relation* R *such that the language* $L_R \notin \mathcal{BPP}$.

Our proofs of the above lower bounds closely follow the methodology and techniques used in previous black-box lower bounds [25, 13].

5.1. Outline of the proofs. As we have mentioned above, the underlying idea behind the proofs of both Theorems 5.2 and 5.1 is the same. We will explain the intuition behind this idea in the context of knowledge extraction (i.e., in the context of the proof of Theorem 5.2) and then describe how this intuition generalizes also to

the context of simulation (for the proof of Theorem 5.1).

Theorem 5.2 is proven by showing that if a language L has a constant-round zero-knowledge protocol (P, V) with a black-box strict polynomial-time knowledge extractor, then there exists a cheating verifier strategy V^*, such that for every $x \in L$, if V^* interacts with the honest prover P, then with noticeable probability, V^* will obtain a witness w for x from the interaction with P. Of course, the ability to do this contradicts the zero-knowledge property of the protocol, unless the verifier V^* could obtain a witness by itself anyway. Since V^* runs in probabilistic polynomial-time, it must therefore be the case that $L \in \mathcal{BPP}$ (because then, indeed, V^* could obtain a witness by itself).

We construct this verifier V^* from the black-box knowledge extractor of the system (P, V). Loosely speaking, this is done in the following way:

1. We let $x \in L$ and consider a cheating prover strategy P^* that is of the following form: P^* behaves exactly as the honest prover P behaves on input x, except that in each round of the proof, it may choose to abort the execution with some probability. We will choose these probabilities so that P^* will still have a noticeable probability to convince the honest verifier to accept x.
   The actual strategy that we use for P^* is one that causes it to abort with quite high probabilities (but still noticeably bounded away from 1). In particular, P^* will abort in each round with probability greater than $1/2$ (and even greater than $1 - 1/n$). Note, however, that since the number of rounds in the protocol is constant, this does not preclude P^* from causing the honest verifier to accept with noticeable probability.
2. Consider an execution of the knowledge extractor when it is given black-box access to P^*. Every query that the extractor makes to the black-box is a list of messages $(\alpha_1, \ldots, \alpha_i)$, and the reply received by the extractor is what P^* would send in a real execution when the first i verifier messages were $\alpha_1, \ldots, \alpha_i$. Thus, if P^* would abort on this series of messages in a real execution, then the extractor receives back an abort reply, denoted by $\perp$. Note that at the end of the execution the knowledge extractor should output a witness for x with some noticeable probability.
   We show that it is possible to choose P^*'s abort probabilities in such a way, that if we run the knowledge extractor with black-box access to P^*, then with very high probability, the extractor will get at most c nonabort replies, where c is the number of verifier messages in the protocol. Furthermore, these replies will correspond to i queries (where $i \leq c$) of the form $(\alpha_1), (\alpha_1, \alpha_2), \ldots, (\alpha_1, \ldots, \alpha_i)$, where $\alpha_1, \ldots, \alpha_i$ are some strings. That is, all these queries are prefixes of a single sequence $(\alpha_1, \ldots, \alpha_i)$. Notice that when this occurs, the extractor's view is like a real interaction with the honest prover P (in particular, it does not gain anything by "rewinding" P^*).
3. We then implement a verifier strategy V^* that sends these messages $\alpha_1, \ldots, \alpha_i$ when it interacts with the prescribed prover P. Basically, the verifier works by internally running the knowledge extractor. When the extractor makes a query, the verifier either answers it with $\perp$ or forwards the query to the prover, and then returns the prover's reply to the knowledge extractor. We show that the verifier has a noticeable probability of perfectly simulating P^* to the knowledge extractor. Therefore, with noticeable probability, the knowledge extractor (in conjunction with the verifier) is able to extract a witness x during this interaction with the prover P. We conclude that with noticeable probability, V^* obtains a witness w for x from the interaction with P, and so

the proof system cannot be zero knowledge (unless $L \in \mathcal{BPP}$).

As mentioned above, in the simulation case (i.e., when proving Theorem 5.1) we use a similar technique. Basically, we prove Theorem 5.1 by showing that if L has a constant-round zero-knowledge proof or argument system (P, V) with a black-box strict polynomial-time simulator, then there exists a cheating prover strategy P^* that does not have any auxiliary input (like a witness), and yet for every $x \in L$ causes the honest verifier V to accept x with noticeable probability. We stress that, unlike the honest prover, this cheating strategy P^* does *not* get a witness for x as auxiliary input. It is not hard to show that the existence of such a strategy P^* implies that $L \in \mathcal{BPP}$.

In summary, both impossibility results are due to the following two facts:

1. Intuitively, in order to successfully extract or simulate, the extractor (resp., simulator) must see at least two different continuations of the same transcript. That is, it must get meaningful replies to queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$, where $\alpha_i' \neq \alpha_i$. Otherwise the extractor (resp., simulator) does not have any advantage over the interactive verifier (resp., prover). (Informally speaking, "rewinding" is essential for black-box simulation and extraction.)
2. For any strict polynomial-time extractor or simulator, there exist provers (resp., verifiers) for which the time required to obtain nonabort responses to two queries $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$, where $\alpha_i' \neq \alpha_i$, is greater than the running time of the extractor (resp., simulator).

Comparison with the black-box zero-knowledge lower bounds of [25, 13]. As we have mentioned above, we use the methodology and techniques of [25, 13] in proving our lower bounds. The lower bounds of [25, 13] and this paper all work by using a black-box simulator to construct a cheating prover P^* that convinces an honest verifier that $x \in L$, without having a witness. The prover P^* works by internally invoking the simulator and forwarding some of the messages between the simulator and the external honest verifier (the other messages are dealt with internally). The main issue to be resolved is how to carry out this execution of the simulator; the difficulty being due to the fact that the simulator is allowed to rewind the verifier during simulation, whereas the cheating prover *cannot* rewind the external honest verifier. Regarding this issue, each of the three lower bounds differs. The lower bound of [25] for constant-round public-coin proofs follows from the fact that in public-coin proofs, all verifier messages are independent of each other. Therefore, it is possible to "guess" messages that should be forwarded to the verifier, and all other messages can be internally generated by the cheating prover (independence of the verifier messages is necessary so that the internal messages generated by the cheating prover do not look inconsistent with messages that the external honest verifier generates). The lower bound of [13] for the concurrent setting is proven by showing that a polynomial-time simulator is unable to rewind the verifier in every execution (because this would require super-polynomial time). Therefore, the messages of the execution in which there is no rewinding can be forwarded to the external real verifier. Finally, in our lower bound, we show that a strict polynomial-time simulator must succeed without effectively rewinding the verifier because rewinding attempts will yield $\perp$ replies with high probability. Therefore, the messages of the simulator that do not yield $\perp$ replies can be sent to the external verifier and no rewinding will be necessary.

5.2. Proof of Theorems 5.2 and 5.1. We now proceed to the actual proofs.

Notation and conventions. We identify an interactive program A with its *next*

message function (or process, if it is randomized). That is, we consider A as a *noninteractive* algorithm that gets as input the history of the interaction (i.e., the sequence of messages that A received until this point), and outputs the next message that A would send in a protocol execution in which it sees this history.

We say that an algorithm B has *oracle access* to an interactive algorithm A, if B has oracle access to A's next message function (after fixing its random tape to some value). That is, B can query its oracle with any sequence of messages of the form $(\alpha_1, \ldots, \alpha_i)$ and it will receive back the next message that A would send in an interaction in which it received this sequence of messages.

We say that an interactive program *aborts* an execution if it sends the message $\perp$. We assume that once a party sends the message $\perp$, then (since it aborted the execution) all its future messages are also $\perp$. In the notation of the next message function, this means that if $A(\alpha_1, \ldots, \alpha_i) = \perp$ for some strings $\alpha_1, \ldots, \alpha_i$, then $A(\alpha_1, \ldots, \alpha_i, \alpha_{i+1}, \ldots, \alpha_j) = \perp$ for every choice of $\alpha_{i+1}, \ldots, \alpha_j$.

The following two lemmas (whose proofs are very similar) lie at the heart of the proofs of Theorems 5.2 and 5.1, respectively.

LEMMA 5.3. *Let (P, V) be a constant-round system of proofs or arguments of knowledge for a relation R with a black-box strict polynomial-time knowledge extractor K. Then, there exists a probabilistic polynomial-time cheating verifier algorithm V^* and a polynomial $p(\cdot)$ such that for every $x \in L_R$, the probability that V^* outputs a witness for x after interacting with the honest prover P on input x is at least $1/p(|x|)$.*

We later show that if the proof system (P, V) is zero knowledge, then the existence of such a cheating verifier V^* implies that $L \in \mathcal{BPP}$. Likewise, we prove the following.

LEMMA 5.4. *Let (P, V) be a constant-round zero-knowledge proof or argument system for a language L, with a black-box strict polynomial-time simulator S. Then, there exists a probabilistic polynomial-time cheating prover algorithm P^* and a polynomial $p(\cdot)$ such that for every $x \in L$, the honest verifier V accepts after interacting with $P^*(x)$ with probability at least $1/p(|x|)$.*

We stress that P^*'s only input is x and, in particular, it does *not* receive a witness for x as auxiliary input. Later we will show that the existence of such a prover P^* implies that $L \in \mathcal{BPP}$ (as shown in previous black-box lower bounds for zero knowledge; e.g., see [25, 13]).

5.2.1. Proof of Lemma 5.3. Let R be a relation and let (P, V) be a system of proofs or arguments of knowledge for R, where the number of messages sent by the verifier equals some constant c. Furthermore, let K be a black-box knowledge extractor for (P, V) that when extracting a witness for x runs for at most $t(n)$ steps, where $n = |x|$ and $t(\cdot)$ is some polynomial. Thus, the number of oracle queries made by K is strictly upper-bounded by a polynomial $t(n)$.[16] We assume without loss of generality that all verifier messages in the proof system are of length $m = m(n)$, where $m(\cdot)$ is some polynomial. We also assume without loss of generality that the first message in the system is from the verifier to the prover.

[16] Note that the running time of the extractor is independent of the running time of the prover. That is, the extractor is restricted to $t(n)$-time (and thus oracle queries) even if the cheating prover runs in time that is larger than $t(n)$. This may seems like an "unfair" restriction, even for a black-box extractor. However, we can extend our lower bound to hold even if the running time of the extractor is allowed to be a fixed polynomial in the running time of the cheating prover. This extension holds under the assumption that one-way functions exist and applies to any proof system that has an efficient prover algorithm. This can be shown by considering a prover strategy P^* that uses a pseudorandom function instead of a t-wise independent hash function, as we use here.

Conventions regarding the extractor K. Recall that a black-box extractor has oracle access to the prover from whom it extracts. Thus it can query this oracle on sequences of messages of the form $(\alpha_1, \ldots, \alpha_i)$. For the sake of simplicity, we can assume without loss of generality that the extractor K always behaves in the following way:

1. It never asks the same query twice.
2. If K queries the oracle with q, then prior to this query, it has queried the oracle with all the proper prefixes of q. (If $q = (\alpha_1, \ldots, \alpha_i)$ is a sequence of messages, then the *prefixes* of q are all the sequences of the form $(\alpha_1, \ldots, \alpha_j)$ for $j \leq i$.)
3. For some polynomial $t' = t'(n)$, the extractor makes exactly $t'(n)$ queries to its black-box in every execution.

We can assume all of the above because any extractor can be easily modified such that it behaves in the above way, without affecting its output distribution. Furthermore, if the upper bound on the number of queries made by the original extractor is $t(n)$, then the number of queries t' made by the modified extractor is at most $c \cdot t(n)$ (the multiplicative factor of c is used for asking all of the prefixes of a query before the query itself). From now on, we will denote the number of queries made by K by $t = t(n)$, rather than by $t'(n)$.

Our proof will follow the outline mentioned at the beginning of section 5. That is, we will construct a prover strategy P^* that aborts with certain probability p^*, but when it does not abort, uses the same strategy as the honest prover. On one hand, the knowledge extractor K will have to output a witness with probability close to p^* when given access to P^*. On the other hand, we will show a cheating verifier V^* that can successfully simulate the behavior of K^{P^*} with noticeable probability, even though it only gets access to one interaction with the honest prover P (and does not get black-box access to P^*).

The prover strategy P^*. We are now ready to describe the prover strategy P^*. Let $\epsilon = \epsilon(n)$ be some value that will be determined later (it will be of the form $1/q(n)$ for some polynomial $q(\cdot)$). Let $\mathcal{H} = \{H_n\}_{n \in \mathsf{N}}$ be a family of $t(n)$-wise independent hash functions,[17] such that every $h \in H_n$ is a function from $\{0,1\}^{\leq c \cdot m}$ to $\{0,1\}^n$, where $\{0,1\}^{\leq c \cdot m}$ denotes the set of all strings of length at most $c \cdot m$. (Recall that $t = t(n)$ denotes the number of queries the knowledge extractor K makes to its black-box.)

Our prover strategy P^* behaves as follows: in the ith round of the protocol, with probability $\epsilon^{2^{c-i}}$, it behaves exactly as the honest prover, and otherwise (with probability $1-\epsilon^{2^{c-i}}$) it aborts. The random coins the prover P^* uses to decide whether to abort or continue will be chosen by applying a fixed hash function $h \in H_n$ (that was initially chosen at random) to the current message history. Before proceeding to a more formal description of P^*, we explain why we choose its abort probabilities in this way. As we have described above, the main idea is to prevent an extractor from obtaining two nonabort responses from P^* for the same round. More exactly, we wish to fix the abort probabilities so that the probability that an extractor sees two nonabort responses for the same round is *significantly less* than the probability that P^* provides a full proof without aborting at all, in which case it convinces V. (Since the extractor must obtain a witness with the probability that P^* convinces V, this means

[17] Recall that a hash family is t-wise independent if for every t distinct values $x_1, \ldots, x_t$ the random variables $h(x_1), \ldots, h(x_t)$ (where h is chosen at random from H_n) are independently and uniformly distributed in the range of h. We will never make more than t queries to the function, and so one can think of it as a truly random function.

that there is a significant probability that the extractor will *not* see two nonabort messages for any given round, but must still succeed in obtaining a witness.) Now, by choosing the abort probabilities so that they degrade exponentially, we achieve our aim. Specifically, for any i, the product $\epsilon^{2^{c-1}} \cdot \ldots \cdot \epsilon^{2^{c-i+1}} \cdot (\epsilon^{2^{c-i}})^2$ (which is the probability that two nonabort responses are obtained for the ith round) is *significantly less* than the product $\epsilon^{2^{c-1}} \cdot \ldots \cdot \epsilon^{2^0}$ (which is the probability that P^* never aborts).

We now describe the strategy for P^*'s operation (note that we define P^* in the form of its next-message function) in the following algorithm:

ALGORITHM 5.5 (prover P^*).

- Common input: x—*the statement to be proven;*
- Auxiliary input: $y \in R(x)$—*a witness for* x*;*
- Random tape: (h, r)—h *defines a function in* H_n*, and* r *is of the length of the random tape required by the honest prover strategy;*
- Input: *series of verifier messages* $q = (\alpha_1, \ldots, \alpha_i)$*.*

1. Step 1—decide whether or not to abort:
   (a) *Compute* $h(q')$ *for every prefix* q' *of* q*. That is, for every* j $(1 \leq j \leq i)$*, compute* $h(\alpha_1, \ldots, \alpha_j)$*.*
   (b) *Abort (outputting a special symbol* $\perp$*), unless for every* j*, the first* $2^{c-j} \log(1/\epsilon)$ *bits of* $h(\alpha_1, \ldots, \alpha_j)$ *are equal to* 0*.*

   (Notice that since the definition of P^* *is by its next message function, we have to ensure that it replies to* q *only if it would not have aborted on messages sent prior to* q *in an interactive setting. This is carried out by checking that it would not have aborted on all prefixes of* q*.)*
2. Step 2—if the decision is to not abort, then follow the honest prover strategy:
   (a) *Run the honest prover* P*, with initial input* (x, y) *and random tape* r*, on input messages* $(\alpha_1, \ldots, \alpha_i)$*, and obtain its response* β*.*
   (b) *Return* β*.*

Note that if the honest prover P is computationally efficient, then so is the cheating prover P^*. (This is important because for the case of arguments, all provers must be efficient.) Our first step is to show that P^* convinces V with noticeable probability. Suppose that the system (P, V) has completeness bound p (i.e., the honest prover causes the honest verifier to accept with probability at least p; recall that p is assumed to be noticeable). Then, define $p^* = \epsilon^{2^c-1} p$. We claim that the probability (over P^*'s random tape) that P^* convinces the honest verifier to accept is at least p^*. Indeed, conditioned on P^* not aborting, its behavior is identical to the behavior of the honest prover P, in which case it convinces V with probability p. Now, since P^*'s probability of not aborting is equal to $\epsilon^{2^{c-1}} \cdot \ldots \cdot \epsilon^{2^0} = \epsilon^{2^c-1}$, the claim follows. As we have mentioned above, we will choose ϵ so that $\epsilon > 1/q(n)$ for some polynomial $q(\cdot)$. Then, since c is a constant, it follows that the probability p^* that P^* convinces the honest V is noticeable, as desired.

We have shown that P^* convinces V with probability at least $p^* = \epsilon^{2^c-1} p$. By the validity (or knowledge soundness) condition on the system (P, V), this means that when the knowledge extractor K is given oracle access to P^*, then it outputs a witness for the statement x with probability at least $p^* - \mu(|x|)$ (where the probability is over the random tapes of both P^* and K).[18] In particular, K outputs a witness

[18] Strictly speaking, the knowledge soundness property is defined for *deterministic* prover strategies. That is, denote by $P^*_{h,r}$ the prover strategy of P^* with its random tape set to (h, r), and denote by $p_{h,r}$ the probability (now over V's random tape only) that $P^*_{h,r}$ convinces V that $x \in L$. Then, the knowledge soundness property states that K must extract a witness from $P^*_{h,r}$ with probability

with probability at least $p^*/2$.

We now claim that if ϵ is chosen appropriately, then with probability at least $p^*/4$, the extractor K will receive at most c non-$\perp$ answers from P^*, and all of these answers are prefixes of a single sequence $(\alpha_1, \ldots, \alpha_i)$ for some $i \leq c$. Informally speaking, this means that with probability at least $p^*/4$, the extractor K is unable to obtain any meaningful information by rewinding P^* (all attempts at rewinding P^* result in abort responses).

CLAIM 5.6. *Let $\tilde{p}$ be the probability (over all choices of P^*'s random tape) that K obtains non-$\perp$ replies for two queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$, where $\alpha_i \neq \alpha_i'$. Then, there is a choice of ϵ such that for every fixed random tape for K, it holds that $\tilde{p} < p^*/4$. Furthermore, $\epsilon = \frac{1}{q(n)}$ for some polynomial $q(\cdot)$.*

Proof. Let $\epsilon(n) = \frac{p}{4 \cdot t^2 c}$ (recall that p is the completeness bound of the honest prover and that t is the number of oracle queries made by K). First, note that for this choice of ϵ, there exists a polynomial $q(\cdot)$ such that $\epsilon = \frac{1}{q(n)}$. Next, for every $i \leq c$ and every $j, k \in [t]$, we let $p^i_{j,k}$ denote the probability that the jth query of K is of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$, the kth query of K is of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$, and K receives a non-$\perp$ response for both of these queries. (Recall that by convention K never asks the same query twice, so this means that $\alpha_i \neq \alpha_i'$.) We now bound the probability $p^i_{j,k}$. Recall that P^* decides whether or not to abort by applying a t-wise independent hash function to the query. By the t-wise independence of the function, P^*'s abort decision is independent for each query. Therefore, if K makes two queries of this form, it will obtain a reply with probability that is $\epsilon^{2^{c-1}+\cdots+2^{c-i+1}} \cdot (\epsilon^{2^{c-i}})^2$, which is the probability that P^* does not abort in the first $i-1$ rounds on history $(\alpha_1, \ldots, \alpha_{i-1})$ multiplied by the probability that P^* does not abort in the ith round both when given the message α_i and when given the message α_i'. Since $2^{c-1}+\cdots+2^{c-i+1}+2 \cdot 2^{c-i} = 2^c$, we have

$$p^i_{j,k} = \epsilon^{2^{c-1}+\cdots+2^{c-i+1}+2 \cdot 2^{c-i}} = \epsilon^{2^c}.$$

By applying the union bound over all $i \in [c]$ and $j, k \in [t]$, we have that $\tilde{p} \leq t^2 c \cdot \epsilon^{2^c}$, where $\tilde{p}$ is defined in the claim statement. Plugging in $p^* = \epsilon^{2^c - 1} p$, we have that $\tilde{p} \leq t^2 c \cdot \epsilon p^*/p$. Finally, since $\epsilon = \frac{p}{4 \cdot t^2 c}$, we have that $\tilde{p} \leq p^*/4$, as required. □

Summing up, we have shown the following. On one hand, when the extractor K is given oracle access to P^*, it must obtain a witness with probability at least $p^*/2$ (where the probability is over the random tapes of both P^*'s and K). On the other hand, the probability that K views two nonabort responses from P^* of the form discussed in Claim 5.6 is at most $p^*/4$. This means that with probability $p^*/2 - p^*/4 = p^*/4$, the extractor K obtains a witness without obtaining non-$\perp$ replies for two queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$, where $\alpha_i \neq \alpha_i'$.

We are now ready to construct the verifier V^*. Loosely speaking, the verifier V^* will manage to output a witness after interacting with the honest prover by internally running the knowledge extractor K, and referring some of its queries to the real prover. The important point is that with noticeable probability, K will work in exactly the same way as when it is given oracle access to P^*. The description of V^* is as follows.

ALGORITHM 5.7 (verifier V^*).

- Input: *x (statement to be proven).*

at least $\tilde{p}_{h,r} = p_{h,r} - \mu(|x|)$. Now, as we have shown, the expectation of $p_{h,r}$ taken over all of P^*'s random tapes is p^* (this is the probability over both P^* and V's random tapes that P^* convinces V that $x \in L$). Therefore, the expectation of $\tilde{p}_{h,r}$ (which is the probability over both P^* and K's random tapes that K extracts a witness for x) is at least $p^* - \mu(|x|)$, as required.

1. *Choose a random $h \in H_n$.*
2. *The verifier will store the history of messages that it has sent to the prover so far in the execution. Initially, this list is empty.*
3. *Run the knowledge extractor K on input x.*
4. *When the extractor K makes a query $(\alpha_1, \ldots, \alpha_k)$ do the following:*
   (a) *Follow the same procedure as P^* in order to decide whether or not to answer the query with $\perp$ (i.e., answer the query with $\perp$ unless for every $j \in [k]$ the first $2^{c-j} log(1/\epsilon)$ bits of $h(\alpha_1, \ldots, \alpha_j)$ are equal to 0).*
   (b) *If the above decision was to not answer the query with $\perp$, then check whether the history of messages sent so far to the prover consists exactly of $(\alpha_1, \ldots, \alpha_{k-1})$. If this is the case, then send α_k to the prover and forward the prover's response β to the knowledge extractor K. If this is not the case, and so there was a previous query that was not answered with $\perp$ and is not a prefix of this query, then abort. (In this case we say that the verifier* fail*.)*
5. *At the end of the execution, if the extractor outputted a witness for x, then output this witness.*

Intuitively, when V^* does not output fail, it perfectly emulates an execution of K with oracle access to P^*. Therefore, V^* outputs a valid witness whenever K would output a witness without obtaining non-$\perp$ replies for two queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$, where $\alpha_i \neq \alpha_i'$. This means that V^* outputs a valid witness with probability at least $p^*/4$.

More formally, let good be the event that K outputs a witness without obtaining non-$\perp$ replies for two queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$, where $\alpha_i \neq \alpha_i'$. Then, as we have shown, the probability that the good event occurs is at least $p^*/4$. Now, the probability space over which we computed the probability that good occurs is (h, r, r_K), where (h, r) is the random tape of P^* and r_K is the random tape of the extractor K. Recall that h was used by P^* for deciding whether or not to abort and r was used by P^* for running the honest prover strategy. Now, observe that the probability space over which we need to compute V^*'s success in outputting a valid witness is also (h, r, r_K). The only difference is that here h and r_K are randomly chosen by V^*, and r is the random tape of the honest prover with which V^* interacts. Therefore the probability space is the same and we have that the probability that V^* outputs a witness is also at least $p^*/4$, which is noticeable. This completes the proof. □

5.2.2. Concluding the proof of Theorem 5.2. To prove Theorem 5.2 we need to show that if (P, V) is a constant-round zero-knowledge proof of knowledge for some relation R with a strict polynomial-time knowledge extractor, then $L_R \in \mathcal{BPP}$. Indeed, if (P, V) is such a system, then by Lemma 5.3, there exists a verifier V^* that, for every $x \in L$, outputs a witness for x with noticeable probability after interacting with the honest prover. Now, since (P, V) is zero knowledge, there exists a simulator S^* for V^* whose output is computationally indistinguishable from the output of V^*. Thus, for every $x \in L_R$, it holds that simulator $S^*(x)$ outputs a witness for x with noticeable probability. On the other hand, if $x \notin L_R$, then S^* will certainly not output a witness for x. Therefore, S^* can be used to obtain a probabilistic polynomial-time procedure for deciding membership in L_R. In other words, $L_R \in \mathcal{BPP}$.

5.2.3. Proof of Lemma 5.4 and Theorem 5.1. The proof of Lemma 5.4 largely follows the proof of Lemma 5.3. Given a c-round zero-knowledge protocol

(P, V) with a black-box strict polynomial-time simulator S, one first constructs a verifier V^* that behaves identically to the honest verifier V, except that it may choose to abort in any round. This verifier V^* corresponds to the cheating prover constructed in Algorithm 5.5 within the proof of Lemma 5.3, and uses the same procedure to decide whether or not to abort. Specifically, the probability that it does not abort in the ith round of the protocol is $\epsilon^{2^{c-i}}$, in which case it follows the instructions of the honest verifier. Now, let $p^* = \epsilon^{2^c-1} \cdot p$, where p is the completeness bound of the zero-knowledge protocol. Then, as above, it follows that an interaction between the honest prover and V^* yields an accepting transcript with probability exactly p^*. Therefore, any simulator given oracle access to V^* must output an accepting transcript with probability at least $p^* - \mu(n) > p^*/2$. Next, a claim analogous to Claim 5.6 is proved. That is, we show that it is possible to choose ϵ so that the probability that the (strict polynomial-time) simulator obtains non-$\perp$ replies from V^* for two queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$, where $\alpha_i \neq \alpha_i'$, is at most $p^*/4$. Furthermore, ϵ is noticeable (implying that p^* is also noticeable). Putting this together, we have that when given black-box access to V^*, the simulator S has a noticeable probability of outputting an accepting transcript, even after viewing at most c non-$\perp$ responses from its oracle, where all these responses are prefixes of the same sequence $(\alpha_1, \ldots, \alpha_i)$. Thus, S can be used to convince the honest verifier with noticeable probability that $x \in L$. That is, in a way similar to the proof of Lemma 5.3, we use this observation about the simulator S in order to construct a cheating prover algorithm P^* (the construction of P^* corresponds to the cheating verifier algorithm described in Algorithm 5.7). This prover P^* does *not* get a witness as auxiliary input, but still for every $x \in L$, it manages to convince the honest verifier to accept with noticeable probability. We omit the full details of the proof of Lemma 5.4.

To prove Theorem 5.1, we need to show that if (P, V) is a constant-round zero-knowledge protocol for L with a black-box strict polynomial-time simulator, then $L \in \mathcal{BPP}$. Now, let (P, V) be such a protocol. By Lemma 5.4, there exists a prover algorithm P^* such that on every input $x \in L$, the prover P^* with input x only (and no witness) manages to convince the honest verifier to accept x with noticeable probability. By the soundness of the system, we know that if $x \notin L$, then the honest verifier will accept x with negligible probability. Therefore, one can test whether or not $x \in L$ in probabilistic polynomial-time, by emulating an interaction between P^* and V on input x, and outputting 1 if and only if the verifier accepts in this execution. Thus, $L \in \mathcal{BPP}$.

Acknowledgments. We wish to thank Oded Goldreich for many helpful discussions and comments. We would also like to thank Moni Naor for pointing out the connection between our lower bound and the separation of black-box ε-knowledge from black-box zero knowledge. Finally, we thank Jonathan Katz for pointing out an error in an earlier version of this work.

REFERENCES

[1] M. Agarwal, N. Kayal, and N. Saxena, *PRIMES is in P*, manuscript, 2002; available from http://www.cse.iitk.ac.in/news/primality.html.

[2] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, *RSA and Rabin functions: Certain parts are as hard as the whole*, SIAM J. Comput., 17 (1988), pp. 194–209.

[3] B. Barak, *How to go beyond the black-box simulation barrier*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, IEEE Computer Soc., Los Alamitos, CA, 2001, pp. 106–115.

[4] B. Barak and O. Goldreich, *Universal arguments and their applications*, in 17th IEEE Conference on Computational Complexity, 2002, pp. 194–203.
[5] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell, *Resettably-sound zero-knowledge and its applications*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, IEEE Computer Soc., Los Alamitos, CA, 2001, pp. 116–125.
[6] M. Bellare and O. Goldreich, *On defining proofs of knowledge*, in CRYPTO '92, Lecture Notes in Comput. Sci. 740, Springer-Verlag, Berlin, 1993, pp. 390–420.
[7] M. Bellare and M. Yung, *Certifying permutations: Non-interactive zero-knowledge based on any trapdoor permutation*, J. Cryptology, 9 (1996), pp. 149–166.
[8] M. Blum, *Coin flipping by phone*, in Proceedings of the 24th IEEE Computer Conference (CompCon), 1982, pp. 133–137; see also SIGACT News, 15 (1983).
[9] M. Blum, *How to prove a theorem so no one else can claim it*, in Proceedings of the International Congress of Mathematicians, Vol. 1, AMS, Providence, RI, 1987, pp. 1444–1451.
[10] G. Brassard, D. Chaum, and C. Crépeau, *Minimum disclosure proofs of knowledge*, J. Comput. System Sci., 37 (1988), pp. 156–189.
[11] G. Brassard, C. Crépeau, and M. Yung, *Everything in NP can be argued in perfect zero-knowledge in a bounded number of rounds*, in EUROCRYPT '89, Lecture Notes in Comput. Sci. 434, Springer-Verlag, Berlin, 1989, pp. 192–195.
[12] S. Goldwasser, R. Canetti, O. Goldreich, and S. Micali, *Resettable zero-knowledge*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 235–244.
[13] R. Canetti, J. Kilian, E. Petrank, and A. Rosen, *Black-box concurrent zero-knowledge requires* $\tilde{\Omega}(\log n)$ *rounds*, SIAM J. Comput., 32 (2002), pp. 1–47.
[14] D. Dolev, C. Dwork, and M. Naor, *Nonmalleable cryptography*, SIAM J. Comput., 30 (2000), pp. 391–437.
[15] C. Dwork, M. Naor, and A. Sahai, *Concurrent zero-knowledge*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 409–418.
[16] U. Feige, D. Lapidot, and A. Shamir, *Multiple noninteractive zero knowledge proofs under general assumptions*, SIAM J. Comput., 29 (1999), pp. 1–28.
[17] U. Feige, *Alternative Models for Zero Knowledge Interactive Proofs*, Ph.D. thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 1990.
[18] U. Feige, A. Fiat, and A. Shamir, *Zero-knowledge proofs of identity*, J. Cryptology, 1 (1988), pp. 77–94.
[19] U. Feige and A. Shamir, *Zero-knowledge proofs of knowledge in two rounds*, in CRYPTO '89, Lecture Notes in Comput. Sci. 435, Springer-Verlag, New York, 1990, pp. 526–544.
[20] O. Goldreich, *Notes on Levin's theory of average-case complexity*, in ECCC: Electronic Colloquium on Computational Complexity, University of Trier, Trier, Germany, 1997, TR97-058.
[21] O. Goldreich, *Secure Multi-party Computation*, manuscript, 2002; available from http://www.wisdom.weizmann.ac.il/∼oded/pp.html.
[22] O. Goldreich, *Foundations of Cryptography: Volume* 1*—Basic Tools*, Cambridge University Press, Cambridge, UK, 2001.
[23] O. Goldreich, *Foundations of Cryptography: Volume* 2*—Basic Applications*, to be published; available from http://www.wisdom.weizmann.ac.il/∼oded/foc-vol2.html.
[24] O. Goldreich and A. Kahan, *How to construct constant-round zero-knowledge proof systems for NP*, J. Cryptology, 9 (1996), pp. 167–189.
[25] O. Goldreich and H. Krawczyk, *On the composition of zero-knowledge proof systems*, SIAM J. Comput., 25 (1996), pp. 169–192.
[26] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity, or All languages in NP have zero-knowledge proof systems*, J. ACM, 38 (1991), pp. 691–729.
[27] O. Goldreich and Y. Oren, *Definitions and properties of zero-knowledge proof systems*, J. Cryptology, 7 (1994), pp. 1–32.
[28] S. Goldwasser and J. Kilian, *Primality testing using elliptic curves*, J. ACM, 46 (1999), pp. 450–472.
[29] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–208.
[30] L. A. Levin, *Average case complete problems*, SIAM J. Comput., 15 (1986), pp. 285–286.
[31] Y. Lindell, *Parallel coin-tossing and constant-round secure two-party computation*, J. Cryptology, 16 (2003), pp. 143–184.
[32] M. Naor, *Bit commitment using pseudorandomness*, J. Cryptology, 4 (1991), pp. 151–158.
[33] M. Tompa and H. Woll, *Random self-reducibility and zero-knowledge interactive proofs of possession of information*, in Proceedings of the 28th IEEE Symposium on Foundations of Computer Science, IEEE Computer Soc., Los Alamitos, CA, 1987, pp. 472–482.

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 819–836

THE COMPLEXITY OF THREE-WAY STATISTICAL TABLES*

JESUS DE LOERA† AND SHMUEL ONN‡

Dedicated to Bernd Sturmfels on the occasion of his 40th birthday

Abstract. Multiway tables with specified marginals arise in a variety of applications in statistics and operations research. We provide a comprehensive complexity classification of three fundamental computational problems on tables: existence, counting, and entry-security.

One outcome of our work is that each of the following problems is intractable already for "slim" 3-tables, with constant number 3 of rows: (1) deciding existence of 3-tables with specified 2-marginals; (2) counting all 3-tables with specified 2-marginals; (3) deciding whether a specified value is attained in a specified entry by at least one of the 3-tables having the same 2-marginals as a given table. This implies that a characterization of feasible marginals for such slim tables, sought by much recent research, is unlikely to exist.

Another consequence of our study is a systematic efficient way of embedding the set of 3-tables satisfying any given 1-marginals and entry upper bounds in a set of slim 3-tables satisfying suitable 2-marginals with no entry bounds. This provides a valuable tool for studying multi-index transportation problems and multi-index transportation polytopes. Remarkably, it enables us to automatically recover a famous example due to Vlach of a "real-feasible integer-infeasible" collection of 2-marginals for 3-tables of smallest possible size $(3, 4, 6)$.

Key words. contingency table, statistical table, data security, statistical disclosure control, Fréchet bound, confidentiality, marginal statistics, data quality, computational complexity, transportation polytope, transportation problems

AMS subject classifications. 68W40, 62H17, 90C27

DOI. 10.1137/S0097539702403803

1. Introduction.

A *d-table of size* $(n_1, \ldots, n_d)$ is an array of nonnegative integers $v = (v_{i_1,\ldots,i_d})$, $1 \le i_j \le n_j$. For $0 \le m < d$, an *m-marginal* of v is any of the $\binom{d}{m}$ possible m-tables obtained by summing the entries over all but m indices. For instance, if $(v_{i,j,k})$ is a 3-table, then its 0-marginal is $v_{+,+,+} = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \sum_{k=1}^{n_3} v_{i,j,k}$, its 1-marginals are $(v_{i,+,+}) = (\sum_{j=1}^{n_2} \sum_{k=1}^{n_3} v_{i,j,k})$, and likewise, $(v_{+,j,+})$, $(v_{+,+,k})$. Its 2-marginals are $(v_{i,j,+}) = (\sum_{k=1}^{n_3} v_{i,j,k})$, and similarly, $(v_{i,+,k})$, $(v_{+,j,k})$.

Such tables appear naturally in statistics and operations research under various names such as *multiway contingency tables*, *transportation matrices*, or *tabular data*. In all these applications, the tables of interest are those satisfying various constraints such as specified marginals or specified upper and lower bounds on the various table entries. Tables are central products of statistical agencies (for example, see the website [2] of the U.S. Bureau of Census).

In this article we study three essential computational problems of constrained tables, primarily motivated by applications in statistical analysis and statistical data security (see, e.g., [10, 12, 14] and the references therein): the *table existence* or *feasibility problem*, the *table counting problem*, and the *table entry-security problem*. We

*Received by the editors March 11, 2002; accepted for publication (in revised form) January 22, 2004; published electronically May 5, 2004.

http://www.siam.org/journals/sicomp/33-4/40380.html

†University of California at Davis, Davis, CA 95616 (deloera@math.ucdavis.edu). The research of this author was supported in part by NSF grants DMS-0073815 and DMS-0309694.

‡Technion, Israel Institute of Technology, 32000, Haifa, Israel (onn@ie.technion.ac.il, http://ie.technion.ac.il/~onn) and University of California at Davis, Davis, CA 95616. The research of this author was supported in part by a grant from the Israel Science Foundation, by a VPR grant at Technion, and by the Technion President Fund.

provide comprehensive computational complexity classifications of these problems, which are discussed, respectively in subsections 1.1–1.3 below, where we make the precise definition, include a briefing on the motivating statistical background, and describe our results for each problem. On the way we also show that the set of 3-tables satisfying any given 1-marginals and upper bounds on entries can be embedded in a set of "slim" 3-tables satisfying suitable 2-marginals with no entry bounds; this is discussed further, with the implications for the class of so-called *multi-index transportation polytopes* (cf. [28, 29]), in subsection 1.4 below.

We demonstrate that even *slim* 3-tables, of size $(3, c, h)$, having a fixed number of rows, 3, can have an arbitrarily complex behavior. This improves on earlier results of Irving and Jerrum [20]. For statisticians and agencies manipulating tabular data, our results have practical repercussions: polynomial time algorithms for solving any one of the problems (feasibility, counting, or entry-security) are unlikely to exist. Thus, in taming 2-marginals arising in practice it will be necessary to exploit particular features of the real data in each specific application.

Our results on the intractability of slim 3-tables stand in contrast with the efficient methods available for 2-way contingency tables (see [16, 17]) and for the so-called decomposable-log-linear models [11, 14, 18]. Thus, we settle a problem that has been the focus of much recent research (see [7, 12, 26] and further references therein), that of trying to find efficient methods for slim 3-tables, and demonstrate that already the 3-tables of smallest possible size $(3, c, h)$ which are not decomposable-graph-log-linear models can have an arbitrarily complex behavior.

Finally, we point out that our results on the intractability of 2-marginals in 3-tables obviously extend to higher dimensions as 2-marginals in 3-tables can be embedded in higher dimensions.

1.1. Table existence or feasibility. First, we consider the *table existence problem*, also called the *feasibility problem* (cf. [5, 28]): *Given a prescribed collection of marginals that seem to describe a d-table of size* $(n_1, \ldots, n_d)$*, does there really exist a table with these marginals?* This problem is relevant for statistical analysis; for instance, disclosed or transmitted marginals may become perturbed or distorted in such a way that a feasible table may no longer exist, in which case not only will the data lose utility to the users, but also algorithms such as the iterative proportional fitting can fail to converge. This can be a problem because several statistical procedures are insensitive to existence, e.g., Fréchet-type bounds presented in [14]. See [5, 6] for a discussion of the importance of table existence in statistics applications. An obvious necessary condition for the existence of a table with a specified collection of marginals is that the collection is *consistent*; that is, any two given marginals must agree on any of their common lower-dimensional marginals. For instance, for the existence of a 3-table with specified 2-marginals $(v_{i,j,+})$ and $(v_{i,+,k})$, these marginals must agree on their common 1-marginal $(v_{i,+,+})$, so the 1-table equation $(\sum_{j=1}^{n} v_{i,j,+}) = (\sum_{k=1}^{n} v_{i,+,k})$ must hold. In general, however, these consistency equations do not even guarantee the existence of an array with nonnegative *real* entries (cf. [28, 29]).

The existence problem is easy to solve for 2-tables [16, 17] or 1-marginals (which are so-called decomposable-log-linear models) [11, 14, 18]. Therefore, the first really interesting case is that of 3-tables with all 2-marginals specified. The following theorem provides an almost complete classification of the complexity of this problem. We assume without loss of generality that the size (r, c, h) of the tables satisfies $3 \leq r \leq c \leq h$. For $r \leq 2$ the problem reduces to the well-studied case of two-dimensional tables (with upper bounds on entries) and is polynomial time solvable using linear

programming over the corresponding *transportation polytope*. See further discussion of these polytopes and their *multi-index* generalizations in subsection 1.4 below.

THEOREM 1.1. *The computational complexity of the existence problem for* 3*-tables of size* (r, c, h) *with* $3 \le r \le c \le h$ *and all* 2*-marginals specified is provided by the following table:*

	r, c, h *fixed*	r, c *fixed,* h *variable*	r *fixed,* c, h *variable*	r, c, h *variable*
unary 2*-marginals*	P	P	NPC	NPC
binary 2*-marginals*	P	?	NPC	NPC

Each entry (i, j) of this table ($i = 1, 2$, $j = 1, 2, 3, 4$) represents a refined version of the existence problem; any problem $(2, j)$ is at least as hard as the problem $(1, j)$ above it, and for $j \ge 2$, any problem (i, j) is at least as hard as the problem $(i, j-1)$ to its left. Here NPC stands for *NP-complete*, and hence presumably intractable and practically unsolvable for large inputs (cf. [15]), whereas P stands for *polynomial time*, and hence efficiently solvable. Binary versus unary are the two standard ways of encoding numbers: for *binary* marginals, the size of each marginal is taken to be the number of digits in its binary (or decimal) expansion, and hence is proportional to its logarithm, whereas for *unary* marginals, the size of each marginal is the marginal itself.

Thus, some of the entries follow at once from others but are included for complete classification. Entry $(1, 1)$ easily follows from an exhaustive search. However, already entry $(2, 1)$ requires the sophisticated algorithm of Lenstra for integer programming in fixed dimension [21]: it would be interesting to devise a special faster polynomial time algorithm for this entry. In section 3 we shall prove entry $(1, 2)$, that is, the polynomial time solvability of existence for unary marginals with r, c fixed ("small") and h variable ("large"). In section 2 we shall establish entries $(1, 3)$ and $(2, 3)$, that is, the NP-completeness of existence for marginals with $r = 3$ fixed and c, h variable. This implies at once entries $(1, 4)$ and $(2, 4)$ in the right column, established previously by Irving and Jerrum [20], strengthening their results. Entry $(2, 2)$ remains unsettled and challenging.

1.2. Table counting. Next, we consider the *table counting problem*: *Given a prescribed collection of marginals, how many d-tables are there that share these marginals?* Table counting has several applications in statistical analysis, in particular independence testing, and has been the focus of much research (see [9, 10, 22] and the extensive list of references therein). The counting problem can be formulated as that of counting the number of integer points in the associated multi-index transportation polytope (see further discussion in subsection 1.4 below). The following analogue of Theorem 1.1 provides a complete classification of the complexity of this problem.

THEOREM 1.2. *The computational complexity of the counting problem for* 3*-tables of size* (r, c, h) *with* $2 \le r \le c \le h$ *and all* 2*-marginals specified is provided by the following table:*

	r, c, h *fixed*	r, c *fixed,* h *variable*	r *fixed,* c, h *variable*	r, c, h *variable*
unary 2*-marginals*	P	P	$\#PC$	$\#PC$
binary 2*-marginals*	P	$\#PC$	$\#PC$	$\#PC$

Here $\#PC$ stands for *#P-complete*, and hence presumably intractable; see Valiant's seminal paper [27], which introduces the complexity theory of counting, or consult [15].

Entry $(1, 1)$ is easy, but already entry $(2, 1)$ requires the sophisticated algorithm of Barvinok for counting integer points in polytopes in fixed dimension [4]. In section 3 we prove entry $(1, 2)$, that is, counting in polynomial time for unary marginals with

r, c fixed and h variable. Entry $(2,2)$ follows from the $\#P$-completeness of counting 2-tables of size $(2,n)$ with binary marginals [13]. In section 3 we also prove entry $(1,3)$, that is, the $\#P$-completeness of counting for unary marginals with $r=2$ fixed and c, h variable, implying entries $(2,3)$, $(1,4)$, and $(2,4)$ as well.

1.3. Table entry-security. The third problem we consider arises in the context of secure and confidential disclosure of public statistical data (see [7, 12, 14] and the references therein). The goal in this context is the release of some marginals of a table in the database but not the table's entries themselves. If the range of possible values that an entry can attain in any table satisfying the released collection of marginals is too narrow, or even worse, consists of the unique value of that entry in the actual table in the database, then this entry may be exposed. This shows the importance of determining tight integer lower and upper bounds on each entry. We consider the following versions of this problem, the *lower* (respectively, *upper*) *table entry-security problem*: *Given a d-table y, a specified collection of marginals of this table, an index tuple $(i_1, \ldots, i_d)$, and a nonnegative integer L (respectively, U), is there a d-table x having the same specified collection of marginals as y, whose entry $x_{i_1,\ldots,i_d}$ is greater than or equal to L (respectively, less than or equal to U)?*

There is an extensive work on the entry-security problem (see, e.g., [6, 11, 14, 25, 26]), where properties are sought that may help address the problem. The state-of-the-art on research and practical techniques is surveyed in [12]. The following analogue of Theorems 1.1 and 1.2 provides an almost complete classification of the complexity of the entry-security problem as well.

THEOREM 1.3. *The complexity of the lower (upper) entry-security problem for 3-tables of size (r,c,h) with $3 \le r \le c \le h$ ($4 \le r \le c \le h$) and all 2-marginals specified is provided by the following table:*

	r,c,h fixed	*r,c fixed, h variable*	*r fixed, c,h variable*	*r,c,h variable*
unary 2-marginals	*P*	*P*	*NPC*	*NPC*
binary 2-marginals	*P*	*?*	*NPC*	*NPC*

Once again, entry $(1,1)$ is easy and entry $(2,1)$ follows from [21]. In section 4 we prove the polynomial time solvability for the case of unary marginals with r, c fixed and h variable (entry $(1,2)$), and the intractability for r fixed and c, h variable (entries $(1,3)$ and $(2,3)$), strengthening earlier hardness results of Irving and Jerrum [20] reflected in entries $(1,4)$ and $(2,4)$. Again, entry $(2,2)$ remains unsettled.

1.4. Multi-index transportation polytopes and the power of 2-marginals. Given a specified collection of marginals for d-tables of size $(n_1, \ldots, n_d)$, possibly together with specified lower and upper bounds on some of the table entries, the associated *multi-index transportation polytope* is the set of all nonnegative *real-valued* arrays satisfying these marginals and entry bounds (cf. [29]) and is a (typically bounded) convex polyhedron in $\mathbf{R}^{n_1 \cdots n_d}$. For instance, for 2-tables of size (n,n) with all 1-marginals equal to 1 and no entry bounds, this is the Birkhoff polytope of n by n bistochastic matrices. The d-tables satisfying the given marginals and entry bounds are precisely the *integer points* in the associated multi-index transportation polytope.

In section 5 we show how a system of 1-marginal and entry upper bound constraints on 3-tables can be embedded into a system of 2-marginal constraints (with no entry bounds) on "slim" 3-tables, demonstrating the expressive power of 2-marginals and reducing the existence, counting, and entry-security problems for 1-marginals with upper bounds to that for 2-marginals with no upper bounds in slim tables. We prove the following somewhat technical statement.

THEOREM 1.4. *Given* 1*-marginals* $(u_{i,+,+})$, $(u_{+,j,+})$, $(u_{+,+,k})$ *and entry upper bounds* $(p_{i,j,k})$ *for* 3*-tables of size* (r,c,h)*, there exist polynomial time constructible* 2*-marginals* $(v_{i,j,+})$, $(v_{i,+,k})$, $(v_{+,j,k})$ *for* 3*-tables of size* $(3, rc, r+c+h)$ *such that the set of nonnegative real* (r,c,h)*-arrays with the given upper bounds and* 1*-marginals is in integer preserving affine bijection with the set of nonnegative real* $(3, rc, r+c+h)$*-arrays with the constructed* 2*-marginals.*

A particularly appealing outcome of our constructions is the systematic and illuminating derivation of "real-feasible integer-infeasible" collections of 2-marginals, admitting nonnegative real 3-arrays but no (integer) 3-tables. Remarkably, applying our construction to very simple $\{0,1\}$-valued 1-marginals and entry upper bounds for 3-tables of size $(2,2,2)$, we "automatically" recover a famous example due to Vlach of a real-feasible integer-infeasible collection of $\{0,1\}$-valued 2-marginals for 3-tables of smallest possible size $(3,4,6)$; see our Example 2.2.

We conclude this introduction with some final discussion. First, we refer to the open problems left in the $(2,2)$ entries of Theorems 1.1 and 1.3. Consider the set of 3-tables v of size (r,c,h) with r,c fixed satisfying specified marginals $(v_{i,+,k})$ and $(v_{+,j,k})$ but *without* restriction on the marginal $(v_{i,j,+})$. The projection $\mathbf{R}^{r\cdot c\cdot h} \longrightarrow \mathbf{R}^{r\cdot c} : v \mapsto (v_{i,j,+}) = \sum_{k=1}^{h} v_{i,j,k}$ sends the associated multi-index transportation polytope onto a subpolytope P of the transportation polytope of all 2-tables of size (r,c) with 1-marginals $(u_{i,+}) := (v_{i,+,+})$ and $(u_{+,j}) := (v_{+,j,+})$. The techniques of [1, 3, 19, 23, 24] allow us to produce the vertices of P in polynomial time and check if any given "vertical" marginal $(v_{i,j,+})$ lies in P, which is necessary for the existence of a 3-table with $(v_{i,+,k})$, $(v_{+,j,k})$, *and* $(v_{i,j,+})$. Further development of the methods of [1, 3, 19, 23, 24] combined with integer programming in fixed dimension might help in addressing these remaining problems.

Although our results stress the complexity of handling even small and slim 3-way tables for statistical applications, recent results using special structure in specific systems may make such table systems amenable to geometric algorithms for practical computations. For example, in [11, 14, 18] the specified marginals satisfy a hierarchical structure of certain graphical models in statistics. Other approaches include the new generation of algebraic and randomized algorithms [8, 13], which will allow, in practice, faster computations for increasingly larger problems.

2. The table existence problem. In this section we provide the proof of Theorem 1.1 discussed in subsection 1.1 and demonstrate our constructions with some examples. In particular, as explained in the introduction, using our construction we recover the smallest possible real-feasible integer-infeasible collection of 2-marginals of 3-tables of size $(3,4,6)$.

Proof of Theorem 1.1. As explained in subsection 1.1, entry $(1,1)$ of the complexity table claimed by Theorem 1.1 is easy and entry $(2,1)$ follows from [21]. Entry $(1,2)$ follows from entry $(1,2)$ in the table of Theorem 1.2, which will be proved in the next section: indeed, we shall show in section 3 how to compute in polynomial time the number of 3-tables of size (r,c,h) with r,c fixed satisfying given 2-marginals in unary, and hence in particular how to decide if this number is zero or not, providing a solution of the existence problem as well.

We need then prove entry $(1,3)$ of the table, which implies at once entries $(2,3),(1,4),(2,4)$ as well. It is easy to see that the well-known three-dimensional matching problem (cf. [15]) is equivalent to the following problem: Given a $\{0,1\}$-valued 3-table $p = (p_{i,j,k})$ of size (n,n,n), is there a 3-table $x = (x_{i,j,k})$ with all 1-marginals equal to 1 which is *dominated* by p, i.e., satisfies the upper bounds

$x_{i,j,k} \le p_{i,j,k}$ for all i,j,k? We reduce this problem to ours, which is clearly in NP. Then let $p = (p_{i,j,k})$ be a given $\{0,1\}$-valued 3-table of size (n,n,n). We define efficiently constructible 2-marginals for $(3, n^2, 3n)$-tables such that nonnegative real arrays y with these marginals are in integer preserving affine bijection with nonnegative real (n,n,n)-arrays x with all 1-marginals equal to 1 dominated by p. For clarity, the table will be indexed by triplets of special form, which we now explain. The first index will be an integer $1 \le t \le 3$. The second index will be an ordered pair ij with $1 \le i,j \le n$. The third index will belong into one of three groups—the "domination" group, the "row" group, and the "column" group—and will consist of one of three three-letter abbreviations $_{\text{gro}} \in \{_{\text{dom}}, _{\text{row}}, _{\text{col}}\}$ according to the group it belongs to, along with a numerical index $1 \le k \le n$. The 2-marginals are provided by the following:

$(v_{+,ij,\text{gro}\,k}) =$

	11	12	$\cdots$	$1n$	21	22	$\cdots$	$2n$	$\cdots$	$n1$	$n2$	$\cdots$	nn
dom 1	$p_{1,1,1}$	$p_{1,2,1}$	$\cdots$	$p_{1,n,1}$	$p_{2,1,1}$	$p_{2,2,1}$	$\cdots$	$p_{2,n,1}$	$\cdots$	$p_{n,1,1}$	$p_{n,2,1}$	$\cdots$	$p_{n,n,1}$
dom 2	$p_{1,1,2}$	$p_{1,2,2}$	$\cdots$	$p_{1,n,2}$	$p_{2,1,2}$	$p_{2,2,2}$	$\cdots$	$p_{2,n,2}$	$\cdots$	$p_{n,1,2}$	$p_{n,2,2}$	$\cdots$	$p_{n,n,2}$
$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
dom n	$p_{1,1,n}$	$p_{1,2,n}$	$\cdots$	$p_{1,n,n}$	$p_{2,1,n}$	$p_{2,2,n}$	$\cdots$	$p_{2,n,n}$	$\cdots$	$p_{n,1,n}$	$p_{n,2,n}$	$\cdots$	$p_{n,n,n}$
row 1	1	1	$\cdots$	1	0	0	$\cdots$	0	$\cdots$	0	0	$\cdots$	0
row 2	0	0	$\cdots$	0	1	1	$\cdots$	1	$\cdots$	0	0	$\cdots$	0
$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
row n	0	0	$\cdots$	0	0	0	$\cdots$	0	$\cdots$	1	1	$\cdots$	1
col 1	1	0	$\cdots$	0	1	0	$\cdots$	0	$\cdots$	1	0	$\cdots$	0
col 2	0	1	$\cdots$	0	0	1	$\cdots$	0	$\cdots$	0	1	$\cdots$	0
$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
col n	0	0	$\cdots$	1	0	0	$\cdots$	1	$\cdots$	0	0	$\cdots$	1

,

$(v_{t,ij,+}) =$

	11	12	$\cdots$	$1n$	21	22	$\cdots$	$2n$	$\cdots$	$n1$	$n2$	$\cdots$	nn
1	1	1	$\cdots$	1	1	1	$\cdots$	1	$\cdots$	1	1	$\cdots$	1
2	$p_{1,1,+}$	$p_{1,2,+}$	$\cdots$	$p_{1,n,+}$	$p_{2,1,+}$	$p_{2,2,+}$	$\cdots$	$p_{2,n,+}$	$\cdots$	$p_{n,1,+}$	$p_{n,2,+}$	$\cdots$	$p_{n,n,+}$
3	1	1	$\cdots$	1	1	1	$\cdots$	1	$\cdots$	1	1	$\cdots$	1

,

$(v_{t,+,\text{gro}\,k}) =$

	1	2	3
dom 1	1	$p_{+,+,1} - 1$	0
dom 2	1	$p_{+,+,2} - 1$	0
$\cdots$	$\vdots$	$\vdots$	$\vdots$
dom n	1	$p_{+,+,n} - 1$	0
row 1	$n-1$	0	1
row 2	$n-1$	0	1
$\cdots$	$\vdots$	$\vdots$	$\vdots$
row n	$n-1$	0	1
col 1	0	1	$n-1$
col 2	0	1	$n-1$
$\cdots$	$\vdots$	$\vdots$	$\vdots$
col n	0	1	$n-1$

.

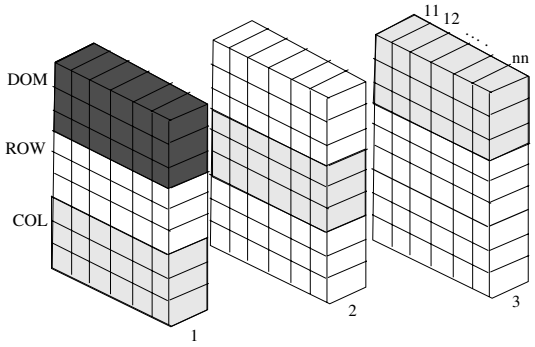


FIG. 1. *Block partition of* $(3, n^2, 3n)$*-arrays.*

First consider any nonnegative real (n, n, n)-array x dominated by p with all 1-marginals equal to 1. We show that it uniquely extends to a nonnegative real $(3, n^2, 3n)$-array with 2-marginals as above. It will be convenient to keep in mind a partition of $(3, n^2, 3n)$-arrays into blocks, as shown in Figure 1. Given then such an array x, embed it in the *black block* $(1, \text{dom})$ of a $(3, n^2, 3n)$-array y by

$$y_{1,ij,\text{dom}\,k} := x_{i,j,k}\,, \qquad 1 \le i, j, k \le n\ .$$

We now show that the block x can be uniquely extended to a whole nonnegative real $(3, n^2, 3n)$-array y with the above 2-marginals. First consider the entries in the *grey blocks* $(1, \text{col})$*,* $(2, \text{row})$*, and* $(3, \text{dom})$ in Figure 1: since $v_{1,+,\text{col}\,k} = v_{2,+,\text{row}\,k} = v_{3,+,\text{dom}\,k} = 0$ for all k, it follows that all the entries $y_{1,ij,\text{col}\,k}$, $y_{2,ij,\text{row}\,k}$, and $y_{3,ij,\text{dom}\,k}$ constituting these blocks are zero. Next, consider the entries in the *white block* $(1, \text{row})$: using the fact just established that all entries in the block $(1, \text{col})$ below it are zero, and examining the 2-marginals $v_{+,ij,\text{row}\,k}$ and $v_{1,ij,+} = 1$, we find that $y_{1,ij,\text{row}\,i} = 1 - \sum_{k=1}^{n} y_{1,ij,\text{dom}\,k} = 1 - x_{i,j,+} \ge 0$, whereas for $k \ne i$ we have $y_{1,ij,\text{row}\,k} = 0$. This also yields the entries in the *white block* $(3, \text{row})$: we have $y_{3,ij,\text{row}\,i} = 1 - y_{1,ij,\text{row}\,i} = x_{i,j,+} \ge 0$, whereas for $k \ne i$ we have $y_{3,ij,\text{row}\,k} = 0$. Next, consider the entries in the *white block* $(2, \text{dom})$: using the fact that all entries in the block $(3, \text{dom})$ to its right are zero, and examining the 2-marginals $v_{+,ij,\text{dom}\,k} = p_{i,j,k}$, we find that $y_{2,ij,\text{dom}\,k} = p_{i,j,k} - x_{i,j,k} \ge 0$ for all i, j, k. Next consider the entries in the *white block* $(2, \text{col})$: using the fact that all entries in the block $(2, \text{row})$ above it are zero, and examining the 2-marginals $v_{+,ij,\text{col}\,k}$ and $v_{2,ij,+} = p_{i,j,+}$, we find that $y_{2,ij,\text{col}\,j} = p_{i,j,+} - \sum_{k=1}^{n} y_{2,ij,\text{dom}\,k} = x_{i,j,+} \ge 0$, whereas for $k \ne j$ we have $y_{2,ij,\text{col}\,k} = 0$. This also yields the entries in the *white block* $(3, \text{col})$: we have $y_{3,ij,\text{col}\,j} = 1 - y_{2,ij,\text{col}\,j} = 1 - x_{i,j,+} \ge 0$, whereas for $k \ne j$ we have $y_{3,ij,\text{col}\,k} = 0$.

Next consider any nonnegative real $(3, n^2, 3n)$-array y with the above 2-marginals, and let x be its (n, n, n)-subarray given by the black block $(1, \text{dom})$ of y, defined by $x_{i,j,k} := y_{1,ij,\text{dom}\,k}$ for all i, j, k. We show that x is nonnegative, dominated by p, and has all 1-marginals equal to 1. It is nonnegative since so is y. It is dominated by p since, for all i, j, k, we have $p_{i,j,k} - x_{i,j,k} = y_{2,ij,\text{dom}\,k} \ge 0$. Finally, all the 1-marginals of x are equal to 1 since

$$x_{+,+,k} \;=\; \sum_{i,j} y_{1,ij,\text{dom}\,k} \;=\; v_{1,+,\text{dom}\,k} \;=\; 1\,, \qquad 1 \le k \le n\ ;$$

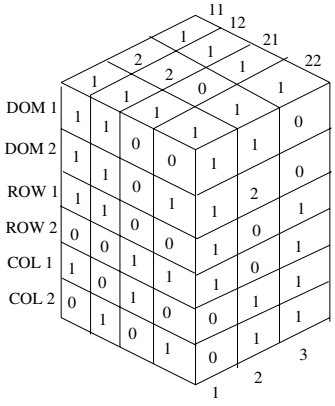


FIG. 2. *The* 2*-marginals* $v_{+,ij,\mathrm{gro}\,k}$, $v_{t,+,\mathrm{gro}\,k}$, $v_{t,ij,+}$ *constructed in Example* 2.1.

$$x_{i,+,+} \;=\; \sum_j y_{3,ij,\mathrm{row}\,i} \;=\; v_{3,+,\mathrm{row}\,i} \;=\; 1\,, \qquad 1 \le i \le n\;;$$

$$x_{+,j,+} \;=\; \sum_i y_{2,ij,\mathrm{col}\,j} \;=\; v_{2,+,\mathrm{col}\,j} \;=\; 1\,, \qquad 1 \le j \le n\;.$$

Thus, the set of nonnegative real (n,n,n)-arrays x dominated by p and with all 1-marginals 1 is in integer preserving affine bijection with the set of nonnegative real $(3,n^2,3n)$-arrays y with the constructed 2-marginals. In particular, the corresponding sets of tables are in bijection, and therefore the former is nonempty if and only if the latter is. This completes the reduction of three-dimensional matching to our problem, the proof of entry $(1,3)$, and the proof of Theorem 1.1.

The following two examples illustrate our construction.

Example 2.1. Let $n = 2$ and let p be the $\{0,1\}$-valued 3-table of size $(2,2,2)$ given by

$$p_{1,1,1}=1,\; p_{1,2,1}=1,\; p_{2,1,1}=0,\; p_{2,2,1}=0,\; p_{1,1,2}=1,\; p_{1,2,2}=1,\; p_{2,1,2}=0,\; p_{2,2,2}=1\,.$$

Our construction yields the 2-marginals for 3-tables of size $(3,4,6)$ presented in Figure 2. The unique 3-table x with all 1-marginals equal to 1 which is dominated by p is given by

$$x_{1,1,1}=1,\; x_{1,2,1}=0,\; x_{2,1,1}=0,\; x_{2,2,1}=0,\; x_{1,1,2}=0,\; x_{1,2,2}=0,\; x_{2,1,2}=0,\; x_{2,2,2}=1\,,$$

and the corresponding 3-table y with the above 2-marginals is given by the following blocks:

$y_{1,ij,\mathrm{gro}\,k}$

	11	12	21	22
dom 1	1	0	0	0
dom 2	0	0	0	1
row 1	0	1	0	0
row 2	0	0	1	0
col 1	0	0	0	0
col 2	0	0	0	0

,

$y_{2,ij,\mathrm{gro}\,k}$

	11	12	21	22
dom 1	0	1	0	0
dom 2	1	1	0	0
row 1	0	0	0	0
row 2	0	0	0	0
col 1	1	0	0	0
col 2	0	0	0	1

,

$y_{3,ij,\mathrm{gro}\,k}$

	11	12	21	22
dom 1	0	0	0	0
dom 2	0	0	0	0
row 1	1	0	0	0
row 2	0	0	0	1
col 1	0	0	1	0
col 2	0	1	0	0

.

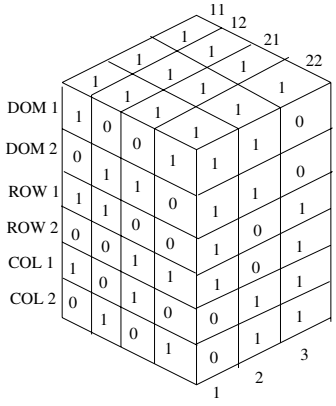


FIG. 3. *Derivation of Vlach's example from our construction.*

As pointed out in the introduction, our construction can be used to systematically obtain "real-nonempty integer-empty" multi-index transportation polytopes, namely, collections of 2-marginals admitting nonnegative real 3-arrays but no (integer) 3-tables. In particular, we next recover the smallest such example, first discovered by Vlach [28], as follows.

Example 2.2. Again, let $n = 2$ and let p be the $\{0,1\}$-valued 3-table of size $(2,2,2)$ given by

$$p_{1,1,1} = 1,\ p_{1,2,1} = 0,\ p_{2,1,1} = 0,\ p_{2,2,1} = 1,\ p_{1,1,2} = 0,\ p_{1,2,2} = 1,\ p_{2,1,2} = 1,\ p_{2,2,2} = 0\,.$$

Our construction yields the 2-marginals for 3-tables of size $(3,4,6)$ presented in Figure 3. It can be verified that there is a single nonnegative real array of size $(2,2,2)$ with all 1-marginals equal to 1 which is dominated by the upper-bound table p. All entries of this array are $\{0,\frac{1}{2}\}$-valued, and there is no (integer) table with the prescribed constraints. Our construction lifts this situation to 2-marginals with no upper bounds: all entries of the unique corresponding nonnegative real array of size $(3,4,6)$ with the 2-marginals in Figure 3 are $\{0,\frac{1}{2}\}$-valued, and there is no (integer) table with these constructed 2-marginals.

3. The table counting problem. In this section we provide the proof of Theorem 1.2 discussed in subsection 1.2.

Proof of Theorem 1.2. As explained in subsection 1.2, entry $(1,1)$ of the complexity table claimed by Theorem 1.2 is easy, entry $(2,1)$ follows from [4], and entry $(2,2)$ follows from [13].

First, we prove entry $(1,3)$ of the table, which implies at once entries $(2,3),(1,4)$, $(2,4)$ as well. We describe a direct reduction from Valiant's canonical $\#P$-complete problem of computing the *permanent* of a $\{0,1\}$-valued matrix [27] (recall that the *permanent* of an n by n matrix A is $\mathrm{perm}(A) := \sum_\sigma \prod_{i=1}^n A_{i,\sigma(i)}$, the sum extending over all permutations σ of $\{1,\ldots,n\}$; for instance, the permanent of the adjacency matrix of a subgraph of the complete bipartite graph $K_{n,n}$ is the number of perfect matchings in that subgraph).

Then let A be a $\{0,1\}$-valued n by n matrix, the permanent of which is to be

computed. Define 2-marginals for 3-tables of size $(2,n,n)$ by

$$v_{i,j,+} := A_{i,j}\,, \quad v_{i,+,1} := v_{+,j,1} := 1\,, \quad v_{i,+,2} := A_{i,+} - 1\,, \quad v_{+,j,2} := A_{+,j} - 1, \qquad 1 \le i,j \le n.$$

Any 3-table x with these marginals is determined by its 2-subtable $(x_{i,j,1})$ since for all i,j we have $x_{i,j,2} = A_{i,j} - x_{i,j,1}$. Now, it is not hard to see that a nonnegative integer n by n matrix Σ can arise as the subtable $(x_{i,j,1})$ of a 3-table x with the constructed 2-marginals if and only if it is the standard representing matrix of a permutation σ satisfying $\prod_{i=1}^{n} A_{i,\sigma(i)} = 1$. Therefore, the permanent of A, which is the number of such permutations σ, is precisely the number of 3-tables with the constructed 2-marginals, completing the reduction and the proof of entry $(1,3)$.

Next we prove entry $(1,2)$ of the table of Theorem 1.2. Note that, as explained in the proof of Theorem 1.1, this implies the corresponding entry $(1,2)$ in the table of Theorem 1.1 as well.

So, r,c are fixed, and we are given unary presented 2-marginals $v_{i,j,+}$, $v_{i,+,k}$, and $v_{+,j,k}$ for 3-tables of size (r,c,h). Let S be the set of all 2-tables s of size (r,c) satisfying the upper bounds $s_{i,j} \le v_{i,j,+}$ for all i,j, that is, dominated by the given "vertical" marginals. For $k = 1,\ldots,h$ define a matrix A_k whose rows and columns are indexed by the elements of S, with entries

$$(A_k)_{s,t} := \begin{cases} 1 & \text{if } (t-s)_{i,+} = v_{i,+,k} \text{ for all } i \text{ and } (t-s)_{+,j} = v_{+,j,k} \text{ for all } j, \\ 0 & \text{otherwise,} \end{cases} \qquad s,t \in S. \tag{3.1}$$

For $p = 1,\ldots,h$ let $A^p := A_1 \cdot A_2 \cdot \ldots \cdot A_p$ be the product of the matrices A_k, $k = 1,\ldots,p$. Further, let l,u denote, respectively, the tables in S with entries $l_{i,j} := 0$ and $u_{i,j} := v_{i,j,+}$ for all i,j.

We claim that for any $1 \le p \le h$ and for any $s,t \in S$, the number of 3-tables x of size (r,c,p) with $x_{i,+,k} = v_{i,+,k}$, $x_{+,j,k} = v_{+,j,k}$, and $x_{i,j,+} = (t-s)_{i,j}$ for $1 \le i \le r$, $1 \le j \le c$, and $1 \le k \le p$ is precisely equal to the entry $A^p_{s,t}$ of A^p. In particular, the number of (r,c,h)-tables with the given 2-marginals is given by $A^h_{l,u}$. Since r,c are fixed and the 2-marginals are presented in unary, the number $\prod_{i=1}^{r}\prod_{j=1}^{c}(v_{i,j,+}+1)$ of tables in S is polynomial in the size of the input, and therefore the matrix A^h and its sought entry $A^h_{l,u}$ can be computed in polynomial time.

We prove the claim by induction on p. First, consider the case $p = 1$ and let s,t be any pair of tables of S. There is a unique $(r,c,1)$-array x satisfying $x_{i,j,1} = x_{i,j,+} = (t-s)_{i,j}$ for all i,j, and x is a table satisfying $x_{i,+,1} = v_{i,+,1}$ and $x_{+,j,1} = v_{+,j,1}$ if and only if $(t-s)_{i,+} = v_{i,+,1}$ and $(t-s)_{+,j} = v_{+,j,1}$ for all i,j, which by (3.1) holds if and only if $A^1_{s,t} = (A_1)_{s,t} = 1$.

Next, consider any $2 \le p \le h$ and suppose the statement is true for all values less than p. Let s,t be any pair of tables of S. Then any (r,c,p)-table x with $x_{i,+,k} = v_{i,+,k}$, $x_{+,j,k} = v_{+,j,k}$, and $x_{i,j,+} = (t-s)_{i,j}$ for all i,j,k is obtained, for some $w \in S$, by augmenting any of the $A^{p-1}_{s,w}$ tables y of size $(r,c,p-1)$ with $y_{i,+,k} = v_{i,+,k}$, $y_{+,j,k} = v_{+,j,k}$, and $y_{i,j,+} = (w-s)_{i,j}$ for all i,j,k by any of the $(A_p)_{w,t}$ tables z of size $(r,c,1)$ with $z_{i,+,1} = v_{i,+,p}$, $z_{+,j,1} = v_{+,j,p}$, and $z_{i,j,+} = (t-w)_{i,j}$ for all i,j. Thus, the number of such tables is $\sum_{w\in S} A^{p-1}_{s,w}(A_p)_{w,t}$, which is precisely $A^p_{s,t}$, proving the induction step and the claim, and thus completing the proof of entry $(1,2)$ and of Theorem 1.2. □

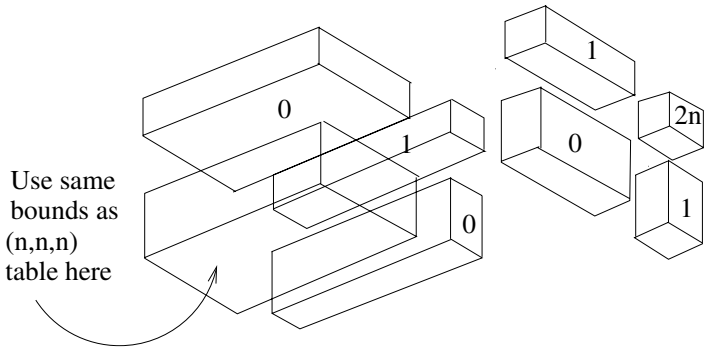


FIG. 4. *The entry bounds for* 3*-tables of size* $(n+1, n+1, n+1)$.

4. The table entry-security problem. In this section we provide the proof of Theorem 1.3 discussed in subsection 1.3.

Proof of Theorem 1.3. Once again, as explained in subsection 1.3, entry $(1,1)$ of the complexity table claimed by Theorem 1.3 is easy and entry $(2,1)$ follows from [21]. We need to prove entry $(1,3)$, which implies at once entries $(2,3), (1,4), (2,4)$ as well, and entry $(1,2)$.

We begin with the proof of entry $(1,3)$: we prove that both the lower and upper entry-security problems are hard. In fact, we prove stronger results by showing that each of the following special cases is already hard: (a) we reduce three-dimensional matching to the problem of deciding whether, given a feasible collection of 2-marginals, there is a 3-table with a specified entry equal to the maximal possible value given by the Fréchet upper bound (minimal value of the three 2-marginals involving this entry). Thus, even the special case of the lower entry-security problem with L the Fréchet upper bound is hard. (b) We reduce the table existence problem to the problem of deciding whether, given a feasible collection of 2-marginals, there is a 3-table with a specified entry equal to the minimal possible value zero; thus, even the special case of the upper entry-security problem with $U = 0$ is hard.

We begin with part (a). We reduce the three-dimensional matching problem to the problem of deciding whether, given feasible 2-marginals, there is a slim 3-table with a specified entry attaining the Fréchet upper bound. As mentioned in section 2, the three-dimensional matching problem is equivalent to the following problem: Given a $\{0,1\}$-valued 3-table $p = (p_{i,j,k})$ of size (n,n,n), is there a 3-table $x = (x_{i,j,k})$ with all 1-marginals $u_{i,+,+}$, $u_{+,j,+}$, $u_{+,+,k}$ equal to 1 which is *dominated* by p, i.e., satisfies the upper bounds $x_{i,j,k} \le p_{i,j,k}$ for all i,j,k? Given such data, we expand it to data for upper bounds and 1-marginals for 3-tables of enlarged size $(n+1, n+1, n+1)$ as follows: we maintain the given upper bounds $p_{i,j,k}$ and the 1-marginals $u_{i,+,+}$, $u_{+,j,+}$, $u_{+,+,k}$ equal to 1 for $1 \le i,j,k \le n$; we introduce the new upper bounds $p_{n+1,n+1,n+1} := 2n$, $p_{i,j,n+1} := p_{i,n+1,k} := p_{n+1,j,k} := 0$ for $1 \le i,j,k \le n$, and $p_{i,n+1,n+1} := p_{n+1,n+1,k} := p_{n+1,j,n+1} := 1$ for $1 \le i,j,k \le n$. Finally, the three new 1-marginals are introduced by $u_{n+1,+,+} := 2n$, $u_{+,n+1,+} := 2n$, and $u_{+,+,n+1} := 2n$. The extended bounds are shown in Figure 4 on the union of the input (n,n,n)-table and seven other blocks. We claim that the extended 1-marginals and upper bounds are feasible: indeed, the $(n+1, n+1, n+1)$-table x defined by setting $x_{i,n+1,n+1} := x_{n+1,n+1,k} := x_{n+1,j,n+1} := 1$ for all $1 \le i,j,k \le n$ and zero in all other entries has the specified marginals.

Now consider any feasible extended table x: then it is not hard to see that its (n,n,n)-subtable $(x_{i,j,k})_{1,1,1}^{n,n,n}$ is feasible for the original data (coming from the input to the three-dimensional matching) if and only if the entry $x_{n+1,n+1,n+1}$ equals the maximal possible value $2n$.

Now, we "lift" the situation to the problem with 2-marginals and no upper bounds in slim tables as follows: To the extended upper bound and 1-marginal data for $(n+1,n+1,n+1)$-tables, apply the transformation described in Theorem 1.4 (to be proved in the next section). This gives feasible 2-marginals for 3-tables of size $(3,(n+1)^2,3(n+1))$. By Theorem 1.4, there is a feasible $(n+1,n+1,n+1)$-table x whose entry $x_{n+1,n+1,n+1}$ attains the maximal possible value $2n$ if and only if there is a feasible $(3,(n+1)^2,3(n+1))$-table y whose entry $y_{1,(n+1)(n+1),\mathrm{dom}\,(n+1)}$ attains the maximal possible value $2n$. This completes the proof of part (a).

Next we prove part (b). Suppose then that we are given all the 2-marginals $v_{i,j,+}$, $v_{+,j,k}$, $v_{i,+,k}$ for 3-tables of size (r,c,h), and we need to decide whether there exists a 3-table with these 2-marginals. For simplicity we denote the given 2-marginals by $A_{i,j} := v_{i,j,+}, B_{j,k} := v_{+,j,k}$, and $C_{i,k} := v_{i,+,k}$. Without loss of generality we may assume that the given 2-marginals are consistent. Recall that consistency means that any pair of adjacent 2-marginals agree on their common 1-marginals and 0-marginals. This is a necessary condition for the problem to be feasible at all. Otherwise there is no table with these marginals. In our situation, consistency means that $A_{i,j}, B_{j,k}$, and $C_{i,k}$ have the same 0-marginal $T := \sum_i^r \sum_k^h C_{i,k} = \sum_i^r \sum_j^c A_{i,j} = \sum_j^c \sum_k^h B_{j,k}$. This is the total entry sum of any 3-table with 2-marginals A, B, C. In addition, any pair of adjacent 2-marginals must agree on their common 1-marginals, i.e., $C_{i,+} = A_{i,+}$, $A_{+,j} = B_{j,+}$, and $C_{+,k} = B_{+,k}$. These consistency equalities will be useful later on.

We will now construct a *feasible* set of 2-marginals for a family of 3-tables $R_{s,t,u}$ of size $(r+1,c+1,h+1)$. The entry-value of a certain entry $R_{s,t,u}$ can be used to decide whether the original set of 2-marginals A,B,C is feasible. We present the 2-marginals in Figure 5 as numbers on the surface of a 3-table. The three 2-marginals $R_{+,t,u}$, $R_{s,+,u}$, $R_{s,t,+}$ are indicated by the coordinate directions in Figure 5.

The reader can verify (see Figure 5) that the assignment is done as follows: for 2-marginal $R_{s,+,u}$ we set $R_{1,+,1} = T$, $R_{1,+,2} = C_{+,h}, R_{1,+,3} = C_{+,h-1}, \ldots, R_{1,+,t} = C_{+,h-t+2}, \ldots, R_{1,+,h+1} = C_{+,1}$. Similarly $R_{2,+,1} = C_{1,+}, \ldots, R_{s,+,1} = C_{s-1,+}, \ldots, R_{r+1,+,1} = C_{r,+}$. Finally, we have the assignment $R_{s,+,u} = C_{s-1,u-1}$ for $s = 2,\ldots,r+1$ and $u = 2,\ldots,h+1$. Next for 2-marginal $R_{s,t,+}$ we have $R_{1,1,+} = T$, $R_{1,2,+} = B_{1,+}, \ldots, R_{1,t,+} = B_{t-1,+}, \ldots, R_{1,c+1,+} = B_{c,+}$; we also have $R_{2,1,+} = C_{1,+}, \ldots, R_{s,1,+} = C_{s-1,+}, \ldots, R_{r+1,1,+} = C_{r,+}$, and $R_{s,t,+} = A_{s-1,t-1}$ for $s = 2,\ldots,r+1$ and $t = 2,\ldots,c+1$. Finally for 2-marginal $R_{+,t,u}$ we have that $R_{+,1,1} = T$ and we set $R_{+,2,1} = A_{+,1}, \ldots, R_{+,t,1} = A_{+,t-1}, \ldots, R_{+,c+1,1} = A_{+,c}$. Also from the picture we see that $R_{+,1,2} = C_{+,h}, \ldots, R_{+,1,u} = C_{+,h-u+2}, \ldots, R_{+,1,h+1} = C_{+,1}$ and $R_{+,t,u} = B_{t-1,u-1}$ for $t = 2,\ldots,c+1$ and $u = 2,\ldots,h+1$.

Note that any such 3-table R with the given 2-marginals breaks up naturally into eight smaller 3-tables. We show these "blocks" $b(1),\ldots,b(8)$ in Figure 6 marking their dimensions. We will use these eight blocks to explain how to fill in the entries of the 3-table and thus to prove that our construction indeed gives (1) *a feasible set of* 2*-marginals and* (2) *the entry* $R_{1,1,1}$ *can be filled in with zero for some* 3*-table satisfying all* 2*-marginals if and only if the original set of* 2*-marginals* A,B,C *is feasible.* The notation we use in subsequent figures to depict a way of filling the entries of a block is by either writing a single number (e.g., zero), which is used to fill *all* the block's entries, or listing a table (e.g., $C_{i,k}$) which indicates that the entries of that table are

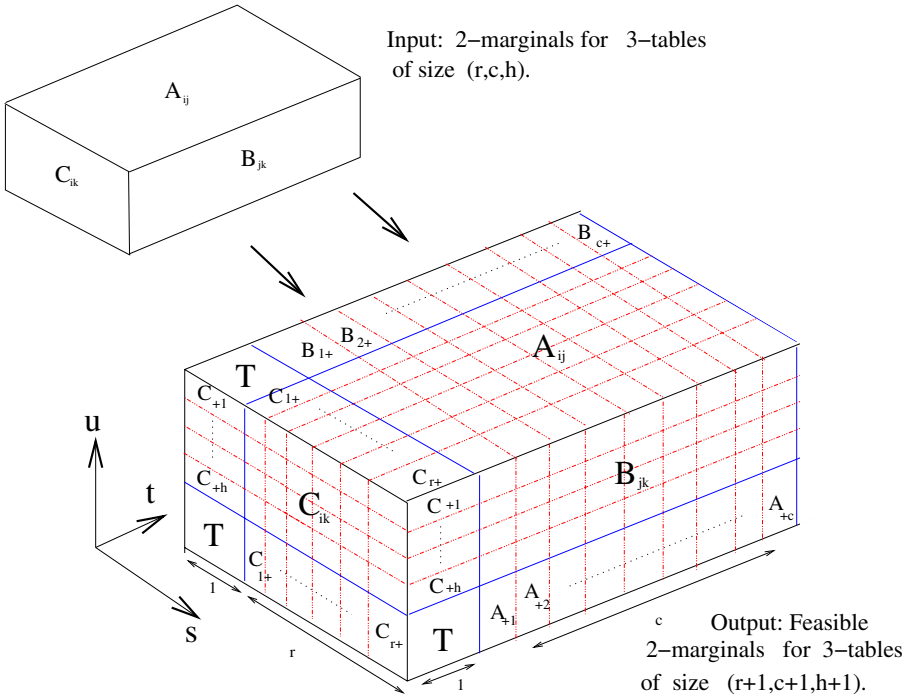


FIG. 5. *The construction of feasible* 2*-marginals from input* 2*-marginals.*

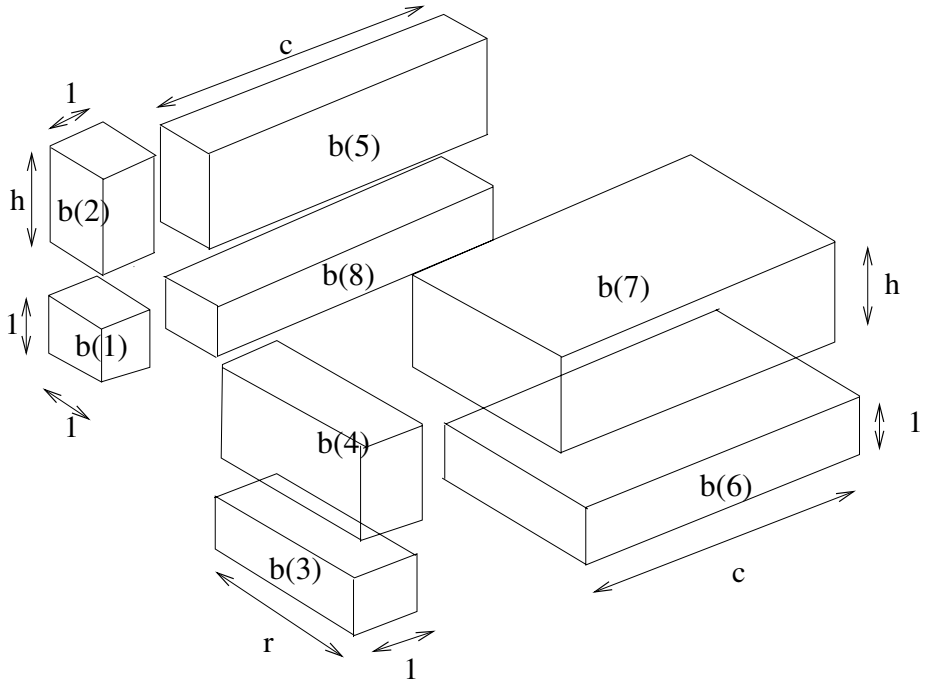


FIG. 6. *Blocks determined by the proposed* 2*-marginals.*

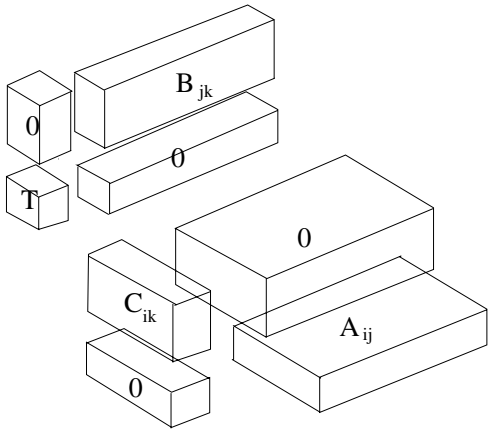


FIG. 7. *A 3-table with the proposed 2-marginals.*

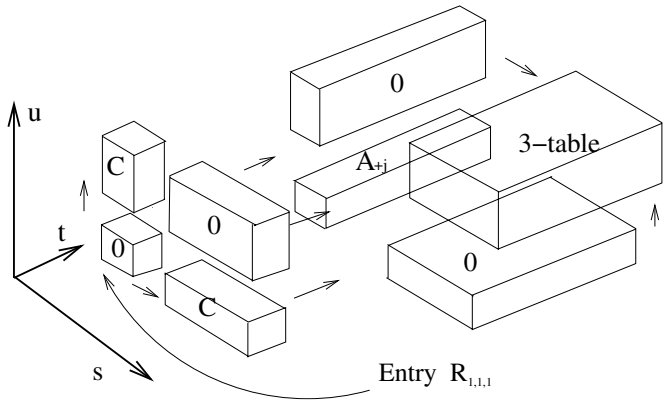


FIG. 8. *A 3-table with the proposed 2-marginals when* $R_{1,1,1} = 0$.

copied down verbatim to be the entries of the block.

Figure 7 shows a concrete 3-table which indeed satisfies the 2-marginals given in the construction. More explicitly, fill the entries as follows: *Block* $b(1)$, a single entry $R_{1,1,1} = T$. *Block* $b(2)$ has entries $R_{1,1,u}$ for $u = 2, \ldots, h+1$. We fill them by $R_{1,1,u} = 0$. *Block* $b(3)$ has entries $R_{s,1,1}$ for $s = 2, \ldots, r+1$. We fill them by $R_{s,1,1} = 0$. *Block* $b(4)$ has entries $R_{s,1,u}$ for $s = 2, \ldots, r+1$ and $u = 2, \ldots, h+1$. We fill them by $R_{s,1,u} = C_{s-1,u-1}$. *Block* $b(5)$ has entries $R_{1,t,u}$ for $t = 2, \ldots, c+1$ and $u = 2, \ldots, h+1$. We fill them by $R_{1,t,u} = B_{t-1,u-1}$. *Block* $b(6)$ has entries $R_{s,t,1}$ for $s = 2, \ldots, c+1$ and $t = 2, \ldots, r+1$. We fill them by $R_{s,t,1} = A_{s-1,t-1}$. The entries of $b(7)$ and $b(8)$ are all zero. It is simple to verify that all the line sums agree with the totals stated in Figure 5, because in the construction we used the 1-marginals of the 2-tables A, B, C as part of the 2-marginals, and the data is consistent. This proves the first claim.

Now we claim that the entry $R_{1,1,1}$ takes on the value zero for some 3-table $R_{s,t,u}$ of size $(r+1, c+1, h+1)$ if and only if the 2-marginals A, B, C have a feasible solution. Let us assume that there is a 3-table $R_{s,t,u}$ of size $(r+1, c+1, h+1)$ and $R_{1,1,1} = 0$. We divide the argument into two steps illustrated on the left-hand side of Figure 8.

If the entry $R_{1,1,1}$ is zero, we must have filled $R_{2,1,1} = C_{1,+}$, $R_{3,1,1} = C_{2,+}, \ldots,$ $R_{(r+1),1,1} = C_{c,+}$ and $R_{1,1,2} = C_{+,1}, \ldots, R_{1,1,2} = C_{+,2}, \ldots, R_{1,1,(h+1)} = C_{+,h}$. The reason is that the 2-marginals $R_{+,1,1}$ and $R_{1,1,+}$ are equal to the total sum T, and $T = \sum C_{i,+} = \sum C_{+,j}$. This completes the filling of blocks $b(2)$ and $b(3)$. Now, the marginals $R_{s,t,+}$ and $R_{+,t,u}$ attached to $b(4)$ imply that $b(4)$ is simply full of zeros; otherwise we surpass the 2-marginals. This completes the first step of the argument.

For the second step we refer to the right-hand side of Figure 8. The marginal table $R_{s,+,u}$ and the assignments so far for blocks $b(2)$ and $b(3)$ imply that only zero values can be put in the entries of blocks $b(5)$ and $b(6)$; otherwise we surpass $R_{s,+,u}$. Now blocks $b(7), b(8)$ are left to be determined. The 2-marginals corresponding to $b(8)$ indicate the entries of $b(8)$ are $R_{1,2,1} = A_{+,1} = B_{1,+}, \ldots, R_{1,j,1} = A_{+,j} = B_{j,+}, \ldots, R_{1,(c+1),1} = A_{+,c} = B_{c,+}$. Note that these are in fact the 1-marginals that follow from the input 2-marginals A, B, C. Finally, block $b(7)$ is the only block unfilled so far. Looking at the zeros that fill blocks $b(4), b(5)$, and $b(6)$, we see block $b(7)$ is indeed a 3-table with 2-marginals A, B, C, and this ends the proof of the claim. Now conversely, and essentially following a reverse order, if there is a 3-table with 2-marginals A, B, C, we can put a copy of it as block $b(7)$. Then by the corresponding 2-marginals we see $b(4), b(5), b(6)$ are filled with zeros. This forces $R_{2,1,1} = C_{1,+}, R_{3,1,1} = C_{2,+}, \ldots, R_{(r+1),1,1} = C_{c,+}$ and $R_{1,1,2} = C_{+,1}, \ldots, R_{1,1,2} = C_{+,2}, \ldots, R_{1,1,(h+1)} = C_{+,h}$. This is because the 2-marginals $R_{+,1,1}$ and $R_{1,1,+}$ equal the total sum T and the 2-marginals $R_{s,+,u}$. Finally, the entry $R_{1,1,1}$ is forced to be zero. This completes the proof of part (b) and the proof of entry $(1,3)$, and hence also of entries $(2,3), (1,4), (2,4)$ in the table of Theorem 1.3.

Finally, we establish entry $(1,2)$ in the statement of Theorem 1.3: we present a polynomial time algorithm for deciding whether there is a 3-table x with specified 2-marginals whose entry $x_{1,1,1}$ is in the range $L \le x_{1,1,1} \le U$. The lower (respectively, upper) entry-security problem is the special case of this entry-range problem obtained by taking U to be the Fréchet upper bound $U := \min\{v_{1,1,+}, v_{1,+,1}, v_{+,+,1}\}$ (respectively, taking $L := 0$). We use a simple modification of the algorithm for enumeration presented in the proof of Theorem 1.2 in the previous section; using the notation in that proof, we simply need to modify the definition of the first matrix A_1, where for $s, t \in S$, its (s,t)th entry is now redefined to be

$$(A_1)_{s,t} := \begin{cases} 1 & \text{if } (t-s)_{i,+} = v_{i,+,k} \text{ for all } i,\ (t-s)_{+,j} = v_{+,j,k} \text{ for all } j, \\ & \text{and } L \le (t-s)_{1,1} \le U, \\ 0 & \text{otherwise.} \end{cases}$$

The other matrices A_k remain as before. The entry $A^h_{l,u}$ of the product matrix now yields the number of tables with $L \le x_{1,1,1} \le U$ and hence is nonzero if and only if such a table exists. □

5. Multi-index transportation polytopes and the power of 2-marginals. We conclude with the proof of Theorem 1.4 discussed in subsection 1.4.

Proof of Theorem 1.4. The proof is based on an extension of the construction used in the proof of Theorem 1.1. We provide the construction and an abridged form of the argumentation.

Given 1-marginals $(u_{i,+,+})$, $(u_{+,j,+})$, $(u_{+,+,k})$ and entry upper bounds $(p_{i,j,k})$ for 3-tables of size (r, c, h), we define efficiently constructible 2-marginals $(v_{i,j,+})$, $(v_{i,+,k})$, $(v_{+,j,k})$ for 3-tables of size $(3, rc, r + c + h)$ such that nonnegative real arrays y with

these marginals are in integer preserving affine bijection with nonnegative real 3-arrays x of size (r,c,h) satisfying the given 1-marginals and upper bounds, thus providing an isomorphism of the corresponding multi-index transportation polytopes and sets of tables of the two systems.

As in the proof of Theorem 1.1, $(3, rc, r+c+h)$-tables will be indexed by triplets, with the first index an integer $1 \le t \le 3$, the second index an ordered pair ij with $1 \le i \le r$ and $1 \le j \le c$, and the third index a three-letter abbreviation $_{\text{gro}} \in \{_{\text{dom}}, _{\text{row}}, _{\text{col}}\}$ along with a numerical index $1 \le k \le h$. Let U denote the minimal of the two values $\max\{u_{i,+,+} : 1 \le i \le r\}$ and $\max\{u_{+,j,+} : 1 \le j \le c\}$. The 2-marginals are provided by the following three matrices:

$(v_{+,ij,\text{gro}\,k}) =$

	11	12	$\cdots$	1c	21	22	$\cdots$	2c	$\cdots$	r1	r2	$\cdots$	rc
dom 1	$p_{1,1,1}$	$p_{1,2,1}$	$\cdots$	$p_{1,c,1}$	$p_{2,1,1}$	$p_{2,2,1}$	$\cdots$	$p_{2,c,1}$	$\cdots$	$p_{r,1,1}$	$p_{r,2,1}$	$\cdots$	$p_{r,c,1}$
dom 2	$p_{1,1,2}$	$p_{1,2,2}$	$\cdots$	$p_{1,c,2}$	$p_{2,1,2}$	$p_{2,2,2}$	$\cdots$	$p_{2,c,2}$	$\cdots$	$p_{r,1,2}$	$p_{r,2,2}$	$\cdots$	$p_{r,c,2}$
$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
dom h	$p_{1,1,h}$	$p_{1,2,h}$	$\cdots$	$p_{1,c,h}$	$p_{2,1,h}$	$p_{2,2,h}$	$\cdots$	$p_{2,c,h}$	$\cdots$	$p_{r,1,h}$	$p_{r,2,h}$	$\cdots$	$p_{r,c,h}$
row 1	U	U	$\cdots$	U	0	0	$\cdots$	0	$\cdots$	0	0	$\cdots$	0
row 2	0	0	$\cdots$	0	U	U	$\cdots$	U	$\cdots$	0	0	$\cdots$	0
$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
row r	0	0	$\cdots$	0	0	0	$\cdots$	0	$\cdots$	U	U	$\cdots$	U
col 1	U	0	$\cdots$	0	U	0	$\cdots$	0	$\cdots$	U	0	$\cdots$	0
col 2	0	U	$\cdots$	0	0	U	$\cdots$	0	$\cdots$	0	U	$\cdots$	0
$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
col c	0	0	$\cdots$	U	0	0	$\cdots$	U	$\cdots$	0	0	$\cdots$	U

,

$(v_{t,ij,+}) =$

	11	12	$\cdots$	1c	21	22	$\cdots$	2c	$\cdots$	r1	r2	$\cdots$	rc
1	U	U	$\cdots$	U	U	U	$\cdots$	U	$\cdots$	U	U	$\cdots$	U
2	$p_{1,1,+}$	$p_{1,2,+}$	$\cdots$	$p_{1,c,+}$	$p_{2,1,+}$	$p_{2,2,+}$	$\cdots$	$p_{2,c,+}$	$\cdots$	$p_{r,1,+}$	$p_{r,2,+}$	$\cdots$	$p_{r,c,+}$
3	U	U	$\cdots$	U	U	U	$\cdots$	U	$\cdots$	U	U	$\cdots$	U

,

$(v_{t,+,\text{gro}\,k}) =$

	1	2	3
dom 1	$u_{+,+,1}$	$p_{+,+,1} - u_{+,+,1}$	0
dom 2	$u_{+,+,2}$	$p_{+,+,2} - u_{+,+,2}$	0
$\cdots$	$\vdots$	$\vdots$	$\vdots$
dom h	$u_{+,+,h}$	$p_{+,+,h} - u_{+,+,h}$	0
row 1	$c \cdot U - u_{1,+,+}$	0	$u_{1,+,+}$
row 2	$c \cdot U - u_{2,+,+}$	0	$u_{2,+,+}$
$\cdots$	$\vdots$	$\vdots$	$\vdots$
row r	$c \cdot U - u_{r,+,+}$	0	$u_{r,+,+}$
col 1	0	$u_{+,1,+}$	$r \cdot U - u_{+,1,+}$
col 2	0	$u_{+,2,+}$	$r \cdot U - u_{+,2,+}$
$\cdots$	$\vdots$	$\vdots$	$\vdots$
col c	0	$u_{+,c,+}$	$r \cdot U - u_{+,c,+}$

.

We make use again of a partition of $(3, rc, r+c+h)$-arrays into blocks similar to Figure 1. First consider any nonnegative real (r, c, h)-array x satisfying the given 1-marginals and upper bounds. We show that it uniquely extends to a nonnegative real $(3, rc, r+c+h)$-array with 2-marginals as above. Given such an array x, embed it in the *black block* $(1, \text{dom})$ of a $(3, rc, r+c+h)$-array y by $y_{1,ij,\text{dom}\,k} := x_{i,j,k}$ for all i, j, k. We now show that the block x can be uniquely extended to a whole nonnegative real $(3, rc, r+c+h)$-array y with the above 2-marginals. First, the entries in the *grey blocks* $(1, \text{col})$, $(2, \text{row})$, *and* $(3, \text{dom})$ in Figure 1 are all zero since so are the 2-marginals $v_{1,+,\text{col}\,k} = v_{2,+,\text{row}\,k} = v_{3,+,\text{dom}\,k} = 0$ for all k. Next, consider the entries in the *white block* $(1, \text{row})$: using the fact that all entries in the block $(1, \text{col})$ below it are zero, and examining the 2-marginals $v_{+,ij,\text{row}\,k}$ and $v_{1,ij,+} = U$, we find that $y_{1,ij,\text{row}\,i} = U - \sum_{k=1}^{h} y_{1,ij,\text{dom}\,k} = U - x_{i,j,+} \geq 0$, whereas for $k \neq i$ we have $y_{1,ij,\text{row}\,k} = 0$. This also yields the entries in the *white block* $(3, \text{row})$: we have $y_{3,ij,\text{row}\,i} = U - y_{1,ij,\text{row}\,i} = x_{i,j,+} \geq 0$, whereas for $k \neq i$ we have $y_{3,ij,\text{row}\,k} = 0$. Next, consider the entries in the *white block* $(2, \text{dom})$: using the fact that all entries in the block $(3, \text{dom})$ to its right are zero, and examining the 2-marginals $v_{+,ij,\text{dom}\,k} = p_{i,j,k}$, we find that $y_{2,ij,\text{dom}\,k} = p_{i,j,k} - x_{i,j,k} \geq 0$ for all i, j, k. Next consider the entries in the *white block* $(2, \text{col})$: using the fact that all entries in the block $(2, \text{row})$ above it are zero, and examining the 2-marginals $v_{+,ij,\text{col}\,k}$ and $v_{2,ij,+} = p_{i,j,+}$, we find that $y_{2,ij,\text{col}\,j} = p_{i,j,+} - \sum_{k=1}^{h} y_{2,ij,\text{dom}\,k} = x_{i,j,+} \geq 0$, whereas for $k \neq j$ we have $y_{2,ij,\text{col}\,k} = 0$. This also yields the entries in the *white block* $(3, \text{col})$: we have $y_{3,ij,\text{col}\,j} = U - y_{2,ij,\text{col}\,j} = U - x_{i,j,+} \geq 0$, whereas for $k \neq j$ we have $y_{3,ij,\text{col}\,k} = 0$.

Next consider any nonnegative real $(3, rc, r+c+h)$-array y with the above 2-marginals, and let x be its (r, c, h)-subarray given by the black block $(1, \text{dom})$ of y, defined by $x_{i,j,k} := y_{1,ij,\text{dom}\,k}$ for all i, j, k. We show that x is nonnegative and satisfies the given upper bounds and 1-marginals. It is nonnegative since so is y. It is dominated by p since, for all i, j, k, we have $p_{i,j,k} - x_{i,j,k} = y_{2,ij,\text{dom}\,k} \geq 0$. Finally, it obeys the 1-marginals $u_{i,+,+}$, $u_{+,j,+}$, and $u_{+,+,k}$ since

$$x_{+,+,k} = \sum_{i,j} y_{1,ij,\text{dom}\,k} = v_{1,+,\text{dom}\,k} = u_{+,+,k}\,, \qquad 1 \leq k \leq h\,;$$

$$x_{i,+,+} = \sum_{j} y_{3,ij,\text{row}\,i} = v_{3,+,\text{row}\,i} = u_{i,+,+}\,, \qquad 1 \leq i \leq r\,;$$

$$x_{+,j,+} = \sum_{i} y_{2,ij,\text{col}\,j} = v_{2,+,\text{col}\,j} = u_{+,j,+}\,, \qquad 1 \leq j \leq c\,.$$

Thus, the set of nonnegative real (r, c, h)-arrays x satisfying the given upper bounds and 1-marginals is in integer preserving affine bijection with the set of nonnegative real $(3, rc, r+c+h)$-arrays y with the constructed 2-marginals. In particular, the corresponding multi-index transportation polytopes and sets of tables of the two systems are isomorphic, completing the proof. □

REFERENCES

[1] N. Alon and S. Onn, *Separable partitions*, Discrete Appl. Math., 91 (1999), pp. 39–51.

[2] *American FactFinder*, U.S. Bureau of the Census, http://factfinder.census.gov/servlet/BasicFactsServlet.

[3] S. Aviran and S. Onn, *Momentopes, the complexity of vector partitioning, and Davenport Schinzel sequences*, Discrete Comput. Geom., 27 (2002), pp. 409–417.
[4] A. I. Barvinok, *A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed*, Math. Oper. Res., 19 (1994), pp. 769–779.
[5] L. H. Cox, *Some remarks on research directions in statistical data protection*, in Statistical Data Protection, Proceedings of Eurostat, Luxembourg, 1999, pp. 163–176.
[6] L. H. Cox, *On Properties of Multi-dimensional Statistical Tables*, manuscript, U.S. Environmental Protection Agency, 2000.
[7] L. H. Cox, *Bounding entries in* 3*-dimensional contingency tables*, in Inference Control in Statistical Databases: From Theory to Practice, Lecture Notes in Comput. Sci. 2316, Springer-Verlag, Heidelberg, 2002, pp. 21–33.
[8] J. De Loera and B. Sturmfels, *Algebraic unimodular counting*, Math. Program., 96 (2003), pp. 183–203.
[9] P. Diaconis and B. Efron, *Testing for independence in a two-way table: New interpretations of the chi-square statistics*, Ann. Statist., 13 (1985), pp. 845–874.
[10] P. Diaconis and A. Gangolli, *Rectangular arrays with fixed margins*, in Discrete Probability and Algorithms (Minneapolis, MN, 1993), IMA Vol. Math. Appl. 72, Springer-Verlag, New York, 1995, pp. 15–41.
[11] A. Dobra and S. E. Fienberg, *Bounds for cell entries in contingency tables given marginal totals and decomposable graphs*, Proc. Natl. Acad. Sci. USA, 97 (2000), pp. 11885–11892.
[12] G. T. Duncan, S. E. Fienberg, R. Krishnan, R. Padman, and S. F. Roehrig, *Disclosure limitation methods and information loss for tabular data*, in Confidentiality, Disclosure and Data Access: Theory and Practical Applications for Statistical Agencies, P. Doyle, J. I. Land, J. M. Theeuwes, and L. V. Zayatz, eds., North-Holland, Amsterdam, 2001, pp. 135–166.
[13] M. E. Dyer, R. Kannan, and J. Mount, *Sampling contingency tables*, Random Structures Algorithms, 10 (1997), pp. 487–506.
[14] S. E. Fienberg, *Fréchet and Bonferroni bounds for multi-way tables of counts with applications to disclosure limitation*, in Statistical Data Protection, Proceedings of Eurostat, Lisbon, 1998, pp. 117–129.
[15] M. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.
[16] D. Gusfield, *A graph theoretic approach to statistical data security*, SIAM J. Comput., 17 (1988), pp. 552–571.
[17] D. Gusfield, *Faster Detection of Compromised Data in* 2*-D Tables*, Technical Report CSE-89-30, Computer Science Department, University of California at Davis, 1989.
[18] S. Hoşten and S. Sullivant, *Gröbner bases and polyhedral geometry of reducible and cyclic models*, J. Combin. Theory Ser. A, 100 (2002), pp. 277–301.
[19] F. K. Hwang, S. Onn, and U. G. Rothblum, *A polynomial time algorithm for shaped partition problems*, SIAM J. Optim., 10 (1999), pp. 70–81.
[20] R. W. Irving and M. R. Jerrum, *Three-dimensional statistical data security problems*, SIAM J. Comput., 23 (1994), pp. 170–184.
[21] H. W. Lenstra, *Integer programming with a fixed number of variables*, Math. Oper. Res., 8 (1983), pp. 538–548.
[22] C. R. Mehta and N. R. Patel, *A network algorithm for performing Fisher's exact test in* $r \times c$ *contingency tables*, J. Amer. Statist. Assoc., 78 (1983), pp. 427–434.
[23] S. Onn and U. G. Rothblum, *Convex combinatorial optimization*, Discrete Comput. Geom., to appear.
[24] S. Onn and L. J. Schulman, *The vector partition problem for convex objective functions*, Math. Oper. Res., 26 (2001), pp. 583–590.
[25] S. F. Roehrig, *Disclosure in multi-way tables with cell suppression: Simplex and shuttle solution*, in Proceedings of the Joint Statistical Meetings, American Statistical Association, Alexandria, VA, 1999.
[26] S. F. Roehrig, *The Integer Rounding Property and Bounds on Contingency Tables*, working paper, The Heinz School of Public Policy and Management, Carnegie Mellon University, Pittsburgh, PA, 2001.
[27] L. G. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.
[28] M. Vlach, *Conditions for the existence of solutions of the three-dimensional planar transportation problem*, Discrete Appl. Math., 13 (1986), pp. 61–78.
[29] V. A. Yemelichev, M. M. Kovalev, and M. K. Kravtsov, *Polytopes, Graphs, and Optimisation*, Cambridge University Press, Cambridge, UK, 1984.

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 837–851

ON MULTIDIMENSIONAL PACKING PROBLEMS*

CHANDRA CHEKURI† AND SANJEEV KHANNA‡

Abstract. We study the approximability of multidimensional generalizations of three classical packing problems: multiprocessor scheduling, bin packing, and the knapsack problem. Specifically, we study the *vector scheduling* problem, its dual problem, namely, the *vector bin packing* problem, and a class of *packing integer programs*. The vector scheduling problem is to schedule n d-dimensional tasks on m machines such that the maximum load over all dimensions and all machines is minimized. The vector bin packing problem, on the other hand, seeks to minimize the number of bins needed to schedule all n tasks such that the maximum load on any dimension across all bins is bounded by a fixed quantity, say, 1. Such problems naturally arise when scheduling tasks that have multiple resource requirements. Finally, packing integer programs capture a core problem that directly relates to both vector scheduling and vector bin packing, namely, the problem of packing a maximum number of vectors in a single bin of unit height. We obtain a variety of new algorithmic as well as inapproximability results for these three problems.

Key words. multidimensional packing, vector scheduling, vector bin packing, packing integer programs, multiprocessor scheduling, bin packing, knapsack, approximation algorithms, hardness of approximation, combinatorial optimization

AMS subject classification. 68Q25

DOI. 10.1137/S0097539799356265

1. Introduction.

Multiprocessor scheduling, bin packing, and the knapsack problem are very well studied problems in combinatorial optimization. Their study has had a large impact on the design and analysis of approximation algorithms. All of these problems involve packing items of different sizes into bins of finite capacities. In this work we study *multidimensional* generalizations of these problems where the items to be packed are d-dimensional vectors and bins are d-dimensional objects as well. We obtain a variety of approximability and inapproximability results in the process, significantly improving upon earlier known results for these problems. Some of our results include a polynomial time approximation scheme (PTAS) for the vector scheduling problem when the dimension is fixed, and an approximation algorithm for the vector bin-packing problem that improves a two-decade-old bound. Though our primary motivation is vector scheduling and vector bin packing, an underlying problem that arises is the problem of maximizing the numbers of vectors that can be packed into a bin of fixed size. This is a special case of the multidimensional knapsack problem that is equivalent to packing integer programs (PIPs) [27, 29]. PIPs are an important class of integer programs that capture several NP-hard combinatorial optimization problems including the maximum independent set problem, the disjoint

*Received by the editors May 17, 1999; accepted for publication (in revised form) December 22, 2003; published electronically May 5, 2004. A preliminary version of this paper appeared in the *Proceedings of the* 10*th Annual ACM-SIAM Symposium on Discrete Algorithms*, Baltimore, MD, 1999, pp. 185–194.

http://www.siam.org/journals/sicomp/33-4/35626.html

†Bell Labs, 600-700 Mountain Ave., Murray Hill, NJ 07974 (chekuri@research.bell-labs.com). Most of this work was done during a summer internship at Bell Labs while the author was at Stanford University. Work at Stanford was supported by an IBM Cooperative fellowship, an ARO MURI grant DAAH04-96-1-0007, and NSF Award CCR-9357849.

‡Department of CIS, University of Pennsylvania, Philadelphia, PA 19104 (sanjeev@cis.upenn.edu). Most of this research was done when the author was at Bell Labs, Lucent Technologies. The research of this author was supported in part by an Alfred P. Sloan Research Fellowship and an NSF Career Award CCR-0093117.

paths problem, and hypergraph matchings. The only general technique known for approximating PIPs is to use randomized rounding on the natural LP relaxation [27, 29]. We show here that the approximation guarantee for PIPs, as obtained via randomized rounding, is essentially the best possible unless NP = ZPP.

In addition to their theoretical importance, these problems have several applications such as load balancing, cutting stock, and resource allocation, to name a few. One of our motivations for studying these problems comes from recent interest [12, 13, 14] in multidimensional resource scheduling problems in parallel query optimization. A favored architecture for parallel databases is the so-called shared-nothing environment [4] where the parallel system consists of a set of independent processing units each of which has a set of time-sharable resources such as CPU, one or more disks, network controllers, etc. A task executing on one of these units has requirements from each of these resources and is best described as a multidimensional load vector. However in most work on scheduling, both in theory and practice, it is assumed that the load of a task is described by a single aggregate work measure. This simplification is done typically to reduce the complexity of the scheduling problem. However, for large task systems that are typically encountered in database applications, ignoring the multidimensionality could lead to bad performance. The work in [11, 12, 13, 14] demonstrates the practical effectiveness of the multidimensional approach. One of the basic resource scheduling problems that is considered in the above papers is the problem of scheduling d-dimensional vectors (tasks) on d-dimensional bins (machines) to minimize the maximum load on any dimension (the load on the most loaded resource). Surprisingly, despite the large body of work on approximation algorithms for multiprocessor scheduling and its several variants [15, 23], the authors in [11] had to settle for a naive $(d+1)$-approximation for the d-dimensional vector scheduling problem. Our work here provides a PTAS when d is fixed and an $O(\ln^2 d)$-approximation when d is arbitrary. A similar situation existed for the vector bin packing problem where the best known approximation ratio prior to our work was $(d+\epsilon)$. In this paper, we improve this to obtain a $(1+\epsilon \cdot d + O(\ln \epsilon^{-1}))$-approximation for any fixed $\epsilon > 0$. In what follows, we formally define the problems that we study and provide a detailed description of our results.

1.1. Problem definitions. We start by defining the vector scheduling problem. For a vector x, the quantity $\|x\|_\infty$ denotes the standard ℓ_∞ norm.

DEFINITION 1.1 (vector scheduling (VS)). *We are given a set J of n rational d-dimensional vectors $p_1, \ldots, p_n$ from $[0,\infty)^d$ and a number m. A valid solution is a partition of J into m sets $A_1, \ldots, A_m$. The objective is to minimize* $\max_{1 \le i \le m} \|\bar{A}_i\|_\infty$, *where $\bar{A}_i = \sum_{j \in A_i} p_j$ is the sum of the vectors in A_i.*

DEFINITION 1.2 (vector bin packing (VBP)). *Given a set of n rational vectors $p_1, \ldots, p_n$ from $[0,1]^d$, find a partition of the set into sets $A_1, \ldots, A_m$ such that $\|\bar{A}_i\|_\infty \le 1$ for $1 \le j \le m$. The objective is to minimize m, the size of the partition.*

The following definition of PIPs is from [29]. In the literature this problem is also referred to as the d-dimensional 0-1 knapsack problem [7].

DEFINITION 1.3 (packing integer program (PIP)). *Given $A \in [0,1]^{d \times n}$, $b \in [1,\infty)^d$, and $c \in [0,1]^n$ with $\max_j c_j = 1$, a PIP seeks to maximize $c^T x$ subject to $x \in \{0,1\}^n$ and $Ax \le b$. Furthermore if $A \in \{0,1\}^{d \times n}$, b is assumed to be integral. Finally, B is defined to be* $\min_i b_i$.

The restrictions on A, b, and c in the above definition are without loss of generality: an arbitrary packing problem can be reduced to the above form (see [29]). We are interested in PIPs where $b_i = B$ for $1 \le i \le d$. When $A \in \{0,1\}^{d \times n}$ this

problem is known as the *simple B-matching in hypergraphs* [24]: given a hypergraph with nonnegative edge weights, find a maximum weight collection of edges such that no vertex occurs in more than B of them. When $B = 1$ this is the usual hypergraph matching problem. We note that the maximum *independent set* problem in graphs is a special case of the hypergraph matching problem.

1.2. Related work and our results. All the problems we consider are NP-complete for $d = 1$ (multiprocessor scheduling, bin packing, and the knapsack problem). The dimension of the vectors, d, plays an important role in determining the complexity. We concentrate on two cases, when d is a fixed constant, and when d is part of the input and can be arbitrary. Below is an outline of the various positive and negative results that we obtain for these problems.

Vector scheduling. For the VS problem the best approximation algorithm [13] prior to our work had a ratio of $(d+1)$. When d is a fixed constant (a case of practical interest) we obtain a PTAS, generalizing the result of Hochbaum and Shmoys [19] for multiprocessor scheduling. In addition we obtain a simpler $O(\ln d)$-approximation algorithm that is better than $(d+1)$ for all $d \geq 2$. When d is large we give an $O(\ln^2 d)$-approximation that uses known approximation algorithms for PIPs as a subroutine. We also give a very simple $O(\ln dm / \ln \ln dm)$-approximation. Finally, we show that it is hard to approximate the VS problem to within *any* constant factor when d is arbitrary.

Vector bin packing. The previous best known approximation algorithms for this problem gave a ratio of $(d + \epsilon)$ for any fixed $\epsilon > 0$ [6] and $(d + 7/10)$ [9]; the latter result holds even in an online setting. All the ratios mentioned are asymptotic; that is, there is an additive term depending on d and on ϵ. Karp, Luby, and Marchetti-Spaccamela [22] do a probabilistic analysis and show bounds on the average wastage in the bins. We design an approximation algorithm that for any fixed $\epsilon > 0$, achieves a $(1 + \epsilon \cdot d + O(\ln \epsilon^{-1}))$-approximation in polynomial time, thus improving upon the previous guarantees. One useful corollary of this result is that when d is a fixed constant we can approximate the problem to within a ratio of $O(\ln d)$. When d is arbitrary a simple reduction from the graph coloring problem gives a $d^{\frac{1}{2}-\epsilon}$ hardness for any fixed $\epsilon > 0$ even when vectors are drawn from the set $[0, 1]^d$. Moreover, even when $d = 2$ the problem is APX-hard [31]; this is an interesting departure from the classical bin packing problem ($d = 1$) which exhibits an asymptotic FPTAS.

Packing integer programs. For fixed d there is a PTAS for PIPs [7]. For large d the randomized rounding technique of Raghavan and Thompson [27] yields integral solutions of value $t_1 = \Omega(\text{OPT}/d^{1/B})$ if $A \in [0, 1]^{d \times n}$, and $t_2 = \Omega(\text{OPT}/d^{1/(B+1)})$ if $A \in \{0, 1\}^{d \times n}$. Srinivasan [29] improved these results to obtain solutions of value $\Omega(t_1^{B/(B-1)})$ and $\Omega(t_2^{(B+1)/B})$, respectively (see discussion at the end of section 4.1 concerning when these values are better). Thus the parameter B plays an important role in the approximation ratio achieved, with better ratios obtained as B gets larger (recall that entries in A are upper bounded by 1). It is natural to question if the dependence of the approximation ratio on B could be any better. We show that PIPs are hard to approximate to within a factor of $\Omega(d^{\frac{1}{B+1}-\epsilon})$ for every fixed B, thus establishing that randomized rounding essentially gives the best possible approximation guarantees. Hardness was known only for the case $B = 1$ via a reduction from the maximum independent set problem. We show how this can be amplified to work for larger values of B and then use Haståd's result [18] on the inapproximability of the maximum independent set problem. An interesting aspect of our reduction is that the hardness result holds even when the optimal is restricted to choosing a solution

that satisfies $Ax \leq 1^d$ while the approximation algorithm is required to satisfy only the relaxed constraint of $Ax \leq B^d$.

Table 1.1 summarizes our results. For conciseness, in the table below when we indicate that a problem is c-hard we mean that, unless NP = ZPP, no polynomial time algorithm can approximate it to within a factor of c.

TABLE 1.1
Approximation bounds and inapproximability results for each problem.

Problem	$d = 1$	Constant $d \geq 2$	Arbitrary d
Vector Scheduling	PTAS [19]	$(d+1)$ (folklore) PTAS (this paper)	$(d+1)$ (folklore) $O(\ln^2 d)$ (this paper)
	NP-hard	NP-hard	NP-hard c-hard $\forall\ c > 1$ (this paper)
Vector Bin Packing	AFPTAS[1] [6, 21]	$(d+\epsilon)$ [6] $O(\ln d)$ (this paper)	$(d+\epsilon)$ [6] $1 + \epsilon d + O(\ln \frac{1}{\epsilon})$ (this paper)
	NP-hard	APX-hard [31]	APX-hard [31] $d^{\frac{1}{2}-\epsilon}$-hard (this paper)
Packing Integer Programs	FPTAS [20]	PTAS [7]	$O(d^{\frac{1}{(B+1)}})$ [26, 27, 29]
	NP-hard	NP-hard	$d^{\frac{1}{2}-\epsilon}$-hard for $B = 1$ $d^{\frac{1}{(B+1)}-\epsilon}$-hard $\forall\ B \geq 1$ (this paper)

1.3. Organization. The rest of the paper is organized as follows. Sections 2 and 3 present our approximation algorithms for the vector scheduling problem and the vector bin packing problem, respectively. In section 4 we present our inapproximability results for the three problems.

2. Algorithms for vector scheduling. For any set of vectors (jobs) A, we define $\bar{A}$ to be the vector sum $\sum_{j \in A} p_j$. The quantity $\bar{A}^i$ denotes component i of the vector $\bar{A}$. Throughout this section, we assume without loss of generality that the vectors have been scaled such that the optimal schedule value is 1.

2.1. A PTAS for fixed *d*. Hochbaum and Shmoys [19] gave a PTAS for the multiprocessor scheduling problem (VS problem with $d = 1$) using dual approximation schemes. We now show that a nontrivial generalization of their ideas yields a PTAS for arbitrary but fixed d.

The basic idea used in [19] is a primal-dual approach whereby the scheduling problem is viewed as a bin packing problem. If an optimal solution can pack all jobs with load not exceeding some height h (assume $h = 1$ from here on), then the scheduling problem is to pack all the jobs into m bins (machines) of height 1. The authors then give an algorithm to solve this bin packing problem with bin height relaxed to $(1 + \epsilon)$, where $\epsilon > 0$ is a fixed constant. In order to do so, they classify a job as large or small depending on whether its size is greater than ϵ or not. Only a fixed number (at most $1/\epsilon$) of large jobs can be packed into any bin. The sizes of the large jobs are then rounded up to be one of $O(\ln 1/\epsilon)$ distinct values. Dynamic programming is used to pack the rounded up large jobs into the m bins such that no

[1] AFPTAS denotes an *asymptotic* FPTAS. A problem has an AFPTAS if for any $\epsilon > 0$, there exists a positive integer N_ϵ such that there is a $(1 + \epsilon)$-approximation algorithm that runs in time polynomial in the input size and $1/\epsilon$ for all instances of the problem with optimum value at least N_ϵ.

bin exceeds a height of $(1+\epsilon)$. The small jobs are then greedily packed on top of the large jobs.

We take a similar approach to the problem. Our dual problem is VBP. The primary difficulty in generalizing the above ideas to the case of jobs or vectors of $d \geq 2$ dimensions is the lack of a total order on the "size" of the jobs. It is still possible to classify a vector as large or small depending on its ℓ_∞ norm. However, the scheme of [19], whereby the small jobs are greedily packed on top of the large jobs, does not apply. We need to take into account the interaction between the packing of large and small vectors. In addition, the packing of small vectors is nontrivial. In fact we use a linear programming relaxation and a careful rounding to pack the small jobs. We describe our ideas in detail below. Following the above discussion we will think of machines as d-dimensional bins and the schedule length as bin capacity (height). Given an $\epsilon > 0$ and a guess for the optimal value (that we assume is normalized to 1), we describe an ϵ-relaxed decision procedure A_ϵ that either returns a schedule of height $(1+\epsilon)$ or proves that the guess is incorrect. We can use A_ϵ to do a binary search for the optimal value. Let δ be ϵ/d.

Preprocessing step. Our first idea is to reduce to zero all coordinates of the vectors that are too small relative to the largest coordinate. This allows us to bound the ratio of the largest coordinate to the smallest nonzero coordinate.

LEMMA 2.1. *Let I be an instance of the VS problem. Let I' be a modified instance where we replace each p_j in I with a vector q_j as follows. For each $1 \leq i \leq d$, $q_j^i = p_j^i$ if $p_j^i > \delta \|p_j\|_\infty$ and $q_j^i = 0$ otherwise. Then, replacing the vector q_j by the vector p_j in any valid solution to I' results in a valid solution to I of height at most a factor of $(1+\epsilon)$ that of I'.*

Proof. Let A be a set of vectors in I, and let B be the corresponding set of vectors in I'. Then it follows from the transformation described above that

$$\begin{aligned}
\bar{A}^i &\leq \bar{B}^i + \delta \sum_{j \in B} \|q_j\|_\infty \\
&\leq \bar{B}^i + \delta \sum_{j \in B} \|q_j\|_1 \\
&\leq \bar{B}^i + \delta \|B\|_1 \\
&\leq \bar{B}^i + \delta d \|B\|_\infty \\
&\leq \bar{B}^i + \epsilon \|B\|_\infty.
\end{aligned}$$

Therefore $\|A\|_\infty \leq (1+\epsilon)\|B\|_\infty$. It follows that replacing vectors in I' by those in I' increases the height of the machines by only a $(1+\epsilon)$ factor. □

Large versus small vectors. Assume from here on that we have transformed our instance as described in the above lemma. The second step in the algorithm is to partition the vectors into two sets L and S corresponding to *large* and *small*. L consists of all vectors whose ℓ_∞ norm is greater than δ, and S is the rest of the vectors. The algorithm A_ϵ will have two stages; the first stage packs all the large jobs, and the second stage packs the small jobs. Unlike the case of $d = 1$, the interaction between the two stages has to be taken into account for $d \geq 2$. We show that the interaction can be captured in a compact way as follows. Let $(a_1, a_2, \ldots, a_d)$ be a d-tuple of integers such that $0 \leq a_i \leq \lceil 1/\epsilon \rceil$. We will call each such distinct tuple a *capacity configuration*. There are at most $t = (1 + \lceil 1/\epsilon \rceil)^d$ such configurations. Assume that the t capacity configurations (tuples) are ordered in some way and let a_i^k be the value of coordinate i in tuple k. A capacity configuration approximately describes how a

bin is filled. However, we have m bins. A t-tuple $(m_1, \ldots, m_t)$, where $0 \leq m_i \leq m$ and $\sum_i m_i = m$, is called a *bin configuration* that describes the number of bins of each capacity configuration. The number of possible bin configurations is clearly $O(m^t)$. Since there are only a polynomial number of such configurations for fixed d and ϵ we can "guess" the configuration used by a feasible packing. A packing of vectors in a bin is said to *respect* a capacity configuration $(a_1, \ldots, a_d)$ if the height of the packing in each dimension i is less than ϵa_i. Given a capacity configuration we can define the corresponding *empty capacity configuration* as the tuple obtained by subtracting each entry from $(\lceil 1/\epsilon \rceil + 1)$. For a bin configuration M we denote by $\bar{M}$ the corresponding bin configuration as the one obtained by taking the empty capacity configurations for each of the bins in M.

Overview of the algorithm. The algorithm performs the following steps for each bin configuration M:

(a) decide if vectors in L can be packed respecting M,

(b) decide if vectors in S can be packed respecting $\bar{M}$.

If both steps above succeed for some M, we have a packing of height at most $(1+\epsilon)$. Otherwise we will prove that the assumption that the optimal packing has a height of 1 is false.

Packing the large vectors. The first stage consists of packing the vectors in L. Observe that the smallest nonzero coordinate of the vectors in L is at least δ^2. We partition the interval $[\delta^2, 1]$ into $q = \lceil \frac{2}{\epsilon} \ln \delta^{-1} \rceil$ intervals of the form $(x_0, (1+\epsilon)x_0]$, $(x_1, (1+\epsilon)x_1], \ldots, (x_{q-1}, 1]$, where $x_0 = \delta^2$ and $x_{i+1} = (1+\epsilon)x_i$. We discretize every nonzero coordinate of the vectors in L by rounding the coordinate down to the left end point of the interval in which it falls. Let L' be the resulting set of vectors.

LEMMA 2.2. *Let I' be an instance obtained from the original instance I by rounding vectors in L as described above. Then, replacing each vector in L' by the corresponding vector in L in any solution for I' results in a solution for I of height at most $(1+\epsilon)$ times that of I'.*

Proof. Each coordinate of a vector in L' is at least $(1+\epsilon)^{-1}$ times the coordinate of the corresponding vector in L. The lemma follows trivially. □

Vectors in L' can be classified into one of $s = (1 + \lceil \frac{2}{\epsilon} \ln \delta^{-1} \rceil)^d$ distinct classes. Any packing of the vectors into one bin can be described as a tuple $(k_1, k_2, \ldots, k_s)$, where k_i indicates the number of vectors of the ith class. Note that at most d/δ vectors from L' can be packed in any bin. Therefore $\sum k_i \leq d/\delta$. Thus there are at most $(d/\delta)^s$ *configurations.* A configuration is *feasible* for a capacity configuration if the vectors described by the configuration can be packed without violating the height constraints described by the capacity configuration. Let C_k denote the set of all configurations of the jobs in L' that are feasible for the kth capacity configuration. From our discussion $|C_k| \leq (d/\delta)^s$.

LEMMA 2.3. *Let $M = (m_1, m_2, \ldots, m_t)$ be a bin configuration. There exists an algorithm with running time $O((d/\delta)^s mn^s)$ to decide if there is a packing of the jobs in L' that respects M.*

Proof. We use a simple dynamic programming-based algorithm. Observe that the number of vector classes in L' is at most s. Thus any subset of vectors from L' can be specified by a tuple of size s, and there are $O(n^s)$ distinct tuples. The algorithm orders bins in some arbitrary way and assigns to each bin a capacity configuration from M. For $1 \leq i \leq m$, the algorithm computes all possible subsets of vectors from L' (tuples) that can be packed validly in the first i bins. For each i this information can be maintained in $O(n^s)$ space. Given the tuples for bin i, the tuples for bin $(i+1)$ can be computed in $O(d/\delta)^s$ time per tuple since that is an upper bound on the

number of feasible configurations for any capacity configuration. Thus for each bin i, in $O((d/\delta)^s n^s)$ time, we can compute the tuples that can be packed into the first i bins given the information for bin $(i-1)$. The number of bins is m so we get the required time bound. □

Packing the small vectors. We now describe the second stage, that of packing the vectors in S. For the second stage we write an integer programming formulation and round the resulting LP relaxation to find an approximate feasible solution. Without loss of generality assume that the vectors in S are numbered 1 to $|S|$. The IP formulation has 0-1 variables x_{ij} for $1 \le i \le |S|$ and $1 \le j \le m$. Variable x_{ij} is 1 if p_i is assigned to machine j. Every vector has to be assigned to some machine. This results in the following equation.

$$\sum_j x_{ij} = 1, \qquad 1 \le i \le |S|. \tag{2.1}$$

Given a bin configuration M, we can define for each machine j and dimension k a height bound b_j^k that an assignment should satisfy. Thus we obtain

$$\sum_i p_i^k \cdot x_{ij} \le b_j^k, \qquad 1 \le j \le m,\ 1 \le k \le d. \tag{2.2}$$

In addition we have the integrality constraints, namely, $x_{ij} \in \{0,1\}$. We obtain a linear program by replacing these constraints by

$$x_{ij} \ge 0. \tag{2.3}$$

PROPOSITION 2.4. *Any* basic feasible solution *to the LP defined by* (2.1), (2.2), *and* (2.3) *has at most* $d \cdot m$ *vectors that are assigned fractionally to more than one machine.*

Proof. The number of variables in our LP is $n \cdot m$. The number of nontrivial constraints (those that are other than $x_{ij} \ge 0$) is $(n + d \cdot m)$. From standard polyhedral theory [28] any basic (vertex) solution to our LP has $n \cdot m$ tight constraints. Therefore by a simple counting argument, at most $(n + d \cdot m)$ variables can be strictly positive. Since each vector is assigned to at least one machine, the number of vectors that are fractionally assigned to more than one machine is at most $d \cdot m$. □

We can solve the above linear program in polynomial time and obtain a basic feasible solution. Let S' be the set of vectors that are not assigned integrally to any machine. By the above lemma, $|S'| \le d \cdot m$. We partition the set S' into m sets of at most d vectors each in an *arbitrary* manner and assign the ith set to the ith machine. Since $\|p_j\|_\infty \le \delta = \epsilon/d$ for every $j \in S'$, the above step does not violate the height by more than ϵ in any dimension.

Putting together all the ingredients we obtain our main theorem below.

THEOREM 2.5. *Given any fixed* $\epsilon > 0$, *there is a* $(1+\epsilon)$*-approximation algorithm for the VS problem that runs in* $(nd/\epsilon)^{O(s)}$ *time, where* $s = O((\frac{\ln(d/\epsilon)}{\epsilon})^d)$.

Proof. Given a correct guess for the optimal schedule height it is clear from the description that we obtain a $(1+O(\epsilon))$-approximation. Following the overview of the algorithm we find a packing of vectors in L and S for each choice of bin configuration M. The running time is dominated by the time to pack L. Since there are at most $m^t = O(n^{O(\epsilon^{-d})})$ bin configurations, the running time follows from Lemma 2.3. We can guess the optimal value to within a $(1+\epsilon)$ precision using the estimate provided by a simple $(d+1)$-approximation algorithm described in section 2.2. □

2.2. The general case. We now consider the case when d is arbitrary and present two approximation algorithms for this case. The first algorithm is deterministic and has an approximation ratio that is a function of only d ($O(\ln^2 d)$), while the second algorithm is randomized and achieves an approximation ratio that is a function of both d and m ($O(\ln dm/\ln\ln dm)$). Given a set of positive vectors A we denote by $\mathcal{V}(A)$ the *volume* of A which is the sum of all coordinates of all vectors in A, in other words, the ℓ_1 norm of $\sum_{j\in A} p_j$. We once again assume that the optimal schedule height is 1.

2.2.1. An $O(\ln^2 d)$-approximation. We start by analyzing a simple algorithm which will serve as a base case for our $O(\ln^2 d)$-approximation algorithm. Recall that J is the set of vectors in the input instance. The infinity norm of each of the job vectors is clearly a lower bound on the optimal value. Hence

$$\max_{j\in J} \|p_j\|_\infty \le 1. \tag{2.4}$$

The second lower bound is obtained by using the average volume per dimension:

$$\frac{\mathcal{V}(J)}{m\cdot d} \le 1. \tag{2.5}$$

We can strengthen the above bound by splitting the sum dimension-wise:

$$\max_{i=1}^{d} \frac{\bar{J}^i}{m} \le 1. \tag{2.6}$$

A naive algorithm for our problem is to ignore the multidimensional aspect of the jobs and treat them as one-dimensional vectors of size equal to the sum of their components. The dimensionality of the bins is also ignored. Then one can apply the standard list scheduling algorithm of Graham [16] for multiprocessor scheduling to obtain the following theorem that uses the simple lower bounds developed above.

LEMMA 2.6. *Applying list scheduling on the volumes of the vectors results in a schedule of height at most* $\frac{\mathcal{V}(J)}{m} + \max_{j\in J}\|p_j\|_\infty$. *This yields a* $(d+1)$*-approximation.*

Proof. The upper bound follows from standard analysis of list scheduling. The approximation guarantee follows from the lower bounds in (2.4) and (2.5). ∎

The $O(\ln^2 d)$-approximation algorithm. Before we state the algorithm formally we need a couple of definitions. The following problem is a special case of a general PIP.

DEFINITION 2.7. *Given a set J of n vectors in $[0,1]^d$, the* largest volume packing problem *is the problem of finding a subset S such that $\|\bar{S}\|_\infty \le 1$ and $\mathcal{V}(S)$ is maximized. Let $\mathcal{V}_{\max}$ denote the value of the optimal solution.*

DEFINITION 2.8. *An (α,β)-approximation to the largest volume packing problem is a subset S that satisfies the conditions $\|\bar{S}\|_\infty \le \alpha$ and $\mathcal{V}(S) \ge \beta\mathcal{V}_{\max}$.*

We will typically use the above definition with $\alpha \ge 1$ and $\beta \le 1$.

Algorithm Volume Pack:

1. **repeat** for t stages
   (a) **for** $k = 1$ to m **do**
      i. Find an (α,β)-approximation to the largest volume packing problem with the current set of job vectors.
      ii. Allocate jobs in packing to machine k and remove them.
2. Find a separate schedule for the remaining jobs using naive volume based list scheduling.
3. Combine the two schedules machine by machine in the obvious way.

We now prove several simple lemmas to analyze the performance of the above algorithm.

LEMMA 2.9. *Let $J(i)$ be the set of jobs remaining at the beginning of the ith stage with $J(1) = J$. Let $J_k(i)$ be the set of jobs remaining after machine k has been packed in stage i. Then*

$$\mathcal{V}(J_k(i)) \leq \mathcal{V}(J(i)) \cdot (1 - \beta/m)^k.$$

Proof. We prove the lemma by induction on k. The claim is trivially true for $k = 0$. Suppose the claim is true up to machine k. We show that the claim is true for machine $(k + 1)$. Since all jobs in J can be scheduled on m machines with height 1, it follows that all jobs in $J_k(i)$ can be likewise scheduled. By a simple averaging argument we can infer that there exists a set of jobs in $J_k(i)$ with volume at least $\mathcal{V}(J_k(i))/m$ that can be packed in a machine with height at most 1. Since we obtain a β-approximation to largest volume packing, we pack jobs with a volume of at least $\beta \cdot \mathcal{V}(J_k(i))/m$. Therefore $\mathcal{V}(J_{(k+1)}(i)) \leq \mathcal{V}(J_k(i)) \cdot (1 - \beta/m)$. By our induction hypothesis $\mathcal{V}(J_k(i)) \leq \mathcal{V}(J(i)) \cdot (1 - \beta/m)^k$. The lemma follows. □

COROLLARY 2.10. $\mathcal{V}(J(i)) \leq \mathcal{V}(J)/e^{(i-1)\beta}$.

Proof. From Lemma 2.9, $\mathcal{V}(J(i)) \leq \mathcal{V}(J(i-1)) \cdot (1-\beta/m)^m$. Since $(1-\beta/m)^m \leq e^{-\beta}$, we get the required bound. □

LEMMA 2.11. *Algorithm* Volume Pack *yields a schedule of height at most* $(t \cdot \alpha + \frac{d}{e^{t\beta}} + 1)$.

Proof. Consider the machine that achieves the maximum height in the schedule produced by Volume Pack. Let J_1 and J_2 be the set of jobs allocated to that machine in the packing stage and the list scheduling stage, respectively. From the packing property it is easy to see that the height of machine due to jobs in J_1 is at most $t\alpha$. Let J' be the set of jobs remaining after the t stages of packing. These are scheduled using list scheduling. From Corollary 2.10 we have that

$$\mathcal{V}(J') \leq \mathcal{V}(J)/e^{t\beta}.$$

From Lemma 2.6, the height increase of the machine due to jobs in J_2 is at most

$$\mathcal{V}(J')/m + \max_j \|p_j\|_\infty \leq \frac{d}{e^{t\beta}} \cdot \mathcal{V}(J)/(dm) + 1 \leq \frac{d}{e^{t\beta}} + 1.$$

In the above inequality we are using the fact that the two lower bounds are less than 1, the optimal value. Combining the two equations gives us the desired bound. □

The parameter t in the algorithm can be chosen as a function of α and β to obtain the best ratio. Note that the largest volume packing problem is a special case of a PIP, where c_i is simply the volume of vector i. PIPs have an $(O(\ln d), 1/2)$-approximation via randomized rounding [27, 29] that can be derandomized by techniques from [26]. When d is fixed there is a $(1, 1 - \epsilon)$ approximation [7] that runs in time polynomial in $n^{d/\epsilon}$. These observations imply the following.

THEOREM 2.12. *There is an $O(\ln^2 d)$-approximation algorithm for the VS problem.*

THEOREM 2.13. *There is an $O(\ln d)$-approximation algorithm for the VS problem that runs in time polynomial in n^d.*

2.2.2. An $O(\ln dm/\ln\ln dm)$-approximation. The approximation result of Theorem 2.12 is good when d is small compared to m. However, when d is large we can obtain an $O(\ln dm/\ln\ln dm)$-approximation by a simple randomized algorithm

that assigns each vector independently to a machine chosen uniformly at random from the set of m machines. Theorem 2.15 bounds the performance of this algorithm, referred to as *Random*. We need a version of the standard Chernoff bound on sums of independent random variables.

PROPOSITION 2.14. *Let $X_1, \ldots, X_n$ be independent binary random variables and let $X = \sum_{i=1}^{n} t_i X_i$. Let $T = \max_i t_i$ and $\mu = \mathbf{E}[X]$. Then, for any sufficiently large h, $\mathbf{Pr}\left[X > (1 + c\frac{\ln h}{\ln \ln h})(\mu + T)\right] \leq h^{-c/2}$.*

Proof. Let Y_i be a random variable such that $Y_i = \frac{t_i}{T} X_i$. Note that Y_i takes on values in $[0, 1]$. Let $Y = \sum_i Y_i$. It follows that $Y = X/T$ and that $\mathbf{E}[Y] = \mu/T$. Therefore, for any $\delta > 0$, $\mathbf{Pr}[X > (1+\delta)(\mu+T)] = \mathbf{Pr}[Y > (1+\delta)(\mu/T + 1)]$. Applying the standard Chernoff–Hoeffding bounds [25] on sums of independent random variables that assume values in $[0, 1]$ to Y, we get the desired result. ∎

THEOREM 2.15. *Random gives an $O(\ln dm / \ln \ln dm)$-approximation with high probability.*

Proof. Consider the first machine. Let X_j be the indicator random variable that is 1 if vector j is assigned to the first machine. The X_j's are independent. By uniformity $\mathbf{Pr}[X_j = 1] = 1/m$. Let $P = \sum_j p_j X_j$. Note that P is a vector since each p_j is a vector: let P^i denote the ith coordinate of P. By linearity of expectations, $\mathbf{E}[P^i] = \sum_j p_j^i / m \leq 1$ (using (2.5)). Also observe that $\max_j p_j^i \leq 1$ (using (2.4)). Now we estimate the probability that P^i deviates significantly from its expected value. From Proposition 2.14, $\mathbf{Pr}\left[P^i > (\mathbf{E}[P^i] + \max_j p_j^i)(1 + c \ln dm / \ln \ln dm)\right] \leq (dm)^{-c/2}$. Thus with high probability P^i is $O(\ln dm / \ln \ln dm)$. In general, if A_k^i is the event that the ith dimension of machine k is greater than $2(1+c) \ln dm / \ln \ln dm$, then from above we know that $\mathbf{Pr}\left[A_k^i\right] \leq (dm)^{-c/2}$. Thus $\mathbf{Pr}\left[(A = \cup_{i=1}^{d} \cup_{k=1}^{m} A_k^i)\right] \leq dm(dm)^{-c/2}$. By choosing c sufficiently large we can ensure that $\mathbf{Pr}[A]$ is less than an inverse polynomial factor. But the complement of A is the event that the schedule length is $O(\ln dm / \ln \ln dm)$. Thus with high probability we get an $O(\ln dm / \ln \ln dm)$-approximation. ∎

3. Algorithms for vector bin packing. We now examine the problem of packing a given set of vectors into the smallest possible number of bins. Our main result here is as follows.

THEOREM 3.1. *For any fixed $\epsilon > 0$, we can obtain in polynomial time a $(1 + \epsilon \cdot d + O(\ln(1/\epsilon)))$-approximate solution for vector bin packing.*

This improves upon the long-standing $(d+\epsilon)$-approximation algorithm of [6]. Our approach is based on solving a linear programming relaxation for this problem. As in section 2.1, we use a variable x_{ij} to indicate if vector p_i is assigned to bin j. We guess the least number of bins m (easily located via binary search) for which the following LP relaxation is feasible; clearly $m \leq \text{OPT}$.

$$\begin{aligned} \sum_j x_{ij} &= 1, && 1 \leq i \leq n, \\ \sum_i p_i^k \cdot x_{ij} &\leq 1, && 1 \leq j \leq m,\ 1 \leq k \leq d, \\ x_{ij} &\geq 0, && 1 \leq i \leq n,\ 1 \leq j \leq m. \end{aligned}$$

Once again, we use the fact that a basic feasible solution would make fractional bin assignments for at most $d \cdot m$ vectors. Thus at this point, all but a set S of at most $d \cdot m$ vectors have integral assignments in m bins. To find a bin assignment for S, we repeatedly find a set $S' \subseteq S$ of up to $k = \lceil 1/\epsilon \rceil$ vectors that can all be packed together

and assign them to a new bin. This step is performed greedily; i.e., we seek to find a largest possible such set in each iteration. We can perform this step by trying out all possible sets of vectors of cardinality less than $(k+1)$. We now claim that this procedure must terminate in $\epsilon \cdot d \cdot m + O(\ln \epsilon^{-1}) \cdot \text{OPT}$ steps. To see this, consider the first time that we pack less than k vectors in a bin. The number of bins used thus far is bounded by $(d \cdot m)/k$. Moreover, the total number of vectors that remain at this point is at most $(k-1)\text{OPT}$; let S' denote this remaining set of vectors. Since the optimal algorithm cannot pack more than $(k-1)$ vectors of S' in one bin, our greedy bin assignment procedure is identical to a greedy set cover algorithm, where each set has size at most $(k-1)$. Following the analysis of the greedy algorithm in [17], the total number of bins used in packing vectors in S' is bounded by $H_{k-1} \cdot \text{OPT}$ (H_i is the ith harmonic number). Putting things together, we obtain that the number of bins used by our algorithm, A, is bounded as follows:

$$A \leq m + (d \cdot m)/k + H_{k-1} \cdot \text{OPT} \leq (1 + \epsilon \cdot d + O(\ln \epsilon^{-1})) \cdot \text{OPT}.$$

This completes the proof of Theorem 3.1. Substituting $\epsilon = 1/d$, we obtain the following simple corollary.

COROLLARY 3.2. *For fixed d, VBP can be approximated to within $O(\ln d)$ in polynomial time.*

4. Inapproximability results. In this section we show hardness of approximation results for the three problems we consider, vector scheduling, vector bin packing, and packing integer programs. We start with PIPs.

4.1. Packing integer programs. Randomized rounding techniques of Raghavan and Thompson [27] yield integral solutions of value $t_1 = \Omega(\text{OPT}/d^{1/B})$ if $A \in [0,1]^{d \times n}$, and $t_2 = \Omega(\text{OPT}/d^{1/(B+1)})$ if $A \in \{0,1\}^{d \times n}$. Srinivasan [29] improved these results to obtain solutions of value $\Omega(t_1^{B/(B-1)})$ and $\Omega(t_2^{(B+1)/B})$, respectively. We show that PIPs are hard to approximate to within a factor of $\Omega(d^{\frac{1}{B+1}-\epsilon})$ for every fixed integer B. We start with the case $A \in \{0,1\}^{d \times n}$ and then indicate how our result extends to $A \in [0,1]^{d \times n}$. Our reduction uses the result of Haståd [18] that shows that unless $NP = ZPP$ the maximum independent set problem is hard to approximate within a factor of $n^{1-\epsilon}$ for any fixed $\epsilon > 0$. Since the upper bounds are in terms of d, from here on, we will express the inapproximability factor only as a function of d.

Given a graph $G = (V, E)$ with $|V| = n$ and a positive integer B, we construct an instance of a PIP, I_G, as follows. Let A be a $d \times n$ 0-1 matrix with $d = n^{(B+1)}$ such that each row corresponds to an element from $V^{(B+1)}$. Let $r_i = (v_{i_1}, \ldots, v_{i_{(B+1)}})$ denote the tuple associated with the ith row of A. We set $a_{ij} = 1$ if and only if the following conditions hold, otherwise we set it to 0: (a) the vertex v_j occurs in r_i, and (b) the vertices in r_i induce a *clique* in G.

We set $c = \{1\}^n$ and $b = \{B\}^d$. For any fixed integer B, the reduction can be done in polynomial time. Note that a feasible solution to I_G can be described as a set of indices $S \subseteq \{1, \ldots, n\}$.

LEMMA 4.1. *Let $X \subseteq V$ be an independent set of G. Then $S = \{i \mid v_i \in X\}$ is a feasible solution to I_G of value $|S| = |X|$. Furthermore, S can be packed with a height bound of 1.*

Proof. Suppose that in some dimension the height induced by S is greater than 1. Let r be the tuple associated with this dimension. Then there exist $i, j \in S$ such that $v_i, v_j \in r$ and $(v_i, v_j) \in E$. This contradicts the assumption that X is an independent set. ∎

LEMMA 4.2. *Let S be any feasible solution to I_G and let G_S be the subgraph of G induced by the set of vertices v_i such that $i \in S$. Then $\omega(G_S) \leq B$, where $\omega(G_S)$ is the clique number of G_S.*

Proof. Suppose there is a clique of size $(B+1)$ in G_S; without loss of generality assume that $v_1, \ldots, v_{(B+1)}$ are the vertices of that clique. Consider the tuple $(v_1, v_2, \ldots, v_{(B+1)})$ and let i be the row of A corresponding to the above tuple. Then by our construction, $a_{ij} = 1$ for $1 \leq j \leq (B+1)$. There are $(B+1)$ vectors in S with a 1 in the same dimension i, violating the ith row constraint. This contradicts the feasibility of S. □

The following is a standard Ramsey type result.

LEMMA 4.3. *Let G be a graph on n vertices with $\omega(G) \leq k$. Then $\alpha(G) \geq n^{1/k}$.*

Lemmas 4.2 and 4.3 give us the following corollary.

COROLLARY 4.4. *Let S be any valid solution to I_G of value $t = |S|$. Then $\alpha(G) \geq t^{1/B}$.*

THEOREM 4.5. *Unless $NP = ZPP$, for every fixed integer B and fixed $\epsilon_0 > 0$, PIPs with bound $b = \{B\}^d$ and $A \in \{0,1\}^{d \times n}$ are hard to approximate to within a factor of $d^{\frac{1}{B+1} - \epsilon_0}$. PIPs with $A \in [0,1]^{d \times n}$ and B rational are hard to approximate to within a factor of $d^{\frac{1}{\lfloor B \rfloor + 1} - \epsilon_0}$.*

Proof. We first look at the case of PIPs with $A \in \{0,1\}^{d \times n}$. Notice that our reduction produces only such instances. Suppose there is a polynomial time approximation algorithm $\mathcal{A}$ for PIPs with bound B that has an approximation ratio $d^{\frac{1}{B+1} - \epsilon_0}$ for some fixed $\epsilon_0 > 0$. This can be reinterpreted as a $d^{\frac{1-\epsilon}{B+1}}$-approximation, where $\epsilon = \epsilon_0(B+1)$ is another constant. We will obtain an approximation algorithm $\mathcal{G}$ for the maximum independent set problem with a ratio $n^{1-\delta}$ for $\delta = \epsilon/B$. The hardness of maximum independent [18] will then imply the desired result. Given a graph G, the algorithm $\mathcal{G}$ constructs an instance I_G of a PIP as described above and gives it as input to $\mathcal{A}$. $\mathcal{G}$ returns $\max(1, t^{1/B})$ as the independent set size of G, where t is the value returned by $\mathcal{A}$ on I_G. Note that by Corollary 4.4, $\alpha(G) \geq t^{1/B}$, which proves the correctness of the algorithm. Now we prove the approximation guarantee. We are interested only in the case when $\alpha(G) \geq n^{1-\delta}$, for otherwise a trivial independent set of size 1 gives the required approximation ratio. From Lemma 4.1 it follows that the optimal value for I_G is at least $\alpha(G)$. Since $\mathcal{A}$ provides a $d^{\frac{1-\epsilon}{B+1}}$-approximation, $t \geq \alpha(G)/d^{\frac{1-\epsilon}{(B+1)}}$. In the construction of I_G, $d = n^{(B+1)}$. Therefore $t \geq \alpha(G)/n^{(1-\epsilon)}$. Simple algebra verifies that $t^{1/B} \geq \alpha(G)/n^{1-\delta}$ when $\alpha(G) \geq n^{1-\delta}$.

Now we consider the case of PIPs with $A \in [0,1]^{d \times n}$. Let B be some real number. For a given B we can create an instance of a PIP as before with $B' = \lfloor B \rfloor$. The only difference is that we set $b = B^d$. Since all entries of A are integral, effectively the bound is B'. Therefore it is hard to approximate to within a factor of $d^{(1-\epsilon)/(B'+1)} = d^{(1-\epsilon)/(\lfloor B \rfloor + 1)}$. Since $(\lfloor B + 1/d \rfloor + 1) = B + 1$, $d^{(1-\epsilon)/(\lfloor B \rfloor + 1)} = \Theta(d^{(1-\epsilon)/B})$. □

Discussion. An interesting aspect of our reduction above is that the hardness results hold even when the optimal algorithm is restricted to a height bound of 1 while allowing a height bound of B for the approximation algorithm. Let an (α, β)-bicriteria approximation be one that satisfies the relaxed constraint matrix $Ax \leq \alpha b$ and gets a solution of value at least OPT/β; here OPT satisfies $Ax \leq b$. Then we have the following corollary.

COROLLARY 4.6. *Unless $NP = ZPP$, for every fixed integer B and fixed $\epsilon > 0$, it is hard to obtain a $(B, d^{\frac{1}{B+1} - \epsilon})$ bicriteria approximation for PIPs.*

For a given B, we use $d = n^{B+1}$, and a hardness of $d^{\frac{1}{B+1} - \epsilon}$ is essentially the

hardness of $n^{1-\epsilon}$ for the independent set problem. This raises two related questions. First, should d be larger than n to obtain the inapproximability results? Second, should the approximability (and inapproximability) results be parameterized in terms of n instead of d? These questions are important to understand the complexity of PIPs as d varies from $O(1)$ to poly(n). We observe that the hardness result holds as long as d is $\Omega(n^{\epsilon})$ for some fixed $\epsilon > 0$. To see this, observe that in our reduction, we can always add poly(n) dummy columns (vectors) that are either useless (their c_j value is 0) or cannot be packed (add a dummy dimension where only B of the dummy vectors can be packed). Thus we can ensure that $n \geq$ poly(d) without changing the essence of the reduction. We have a PTAS when $d = O(1)$ and a hardness result of $d^{1/(B+1)}$ when $d =$ poly(n). An interesting question is to resolve the complexity of the problem when $d =$ polylog(n).

As remarked earlier, Srinivasan [29] improves the results obtained using randomized rounding to obtain solutions of value $\Omega(t_2^{(B+1)/B})$, where $t_2 = \Omega(y^*/d^{1/(B+1)})$ for $A \in \{0,1\}^{d \times n}$. In the above y^* is the optimal fractional solution to the PIP. It might appear that this contradicts our hardness result but observe that for the instances we create in our reduction $y^*/d^{1/(B+1)} \leq 1$. For such instances Srinivasan's bounds do not yield an improvement over randomized rounding.

4.2. Vector scheduling. We now extend the ideas used in the hardness result for PIPs to show the following hardness result for the VS problem.

THEOREM 4.7. *Unless NP = ZPP, for every constant $\gamma > 1$, there is no polynomial time algorithm that approximates the schedule height in the VS problem to within a factor of γ.*

Our result here uses the hardness of graph coloring; Feige and Kilian [5] building on the work of Hastad [18] show that graph coloring is $n^{1-\epsilon}$-hard unless NP = ZPP. Our reduction is motivated by the fact that graph coloring corresponds to covering a graph by independent sets. We start with the following simple lemma that is easily derived from Lemma 4.3.

LEMMA 4.8. *Let G be a graph on n vertices with $\omega(G) \leq k$. Then $\chi(G) \leq O(n^{1-1/k} \ln n)$.*

Let $B = \lceil \gamma \rceil$; we will show that it is hard to obtain a B-approximation using a reduction from chromatic number. Given graph G we construct an instance I of the VS problem as follows. We construct n vectors of n^{B+1} dimensions as in the proof of Theorem 4.5. We set m, the number of machines, to be $n^{\frac{1}{2B}}$.

LEMMA 4.9. *If $\chi(G) \leq m$, then the optimal schedule height for I is 1.*

Proof. Let $V_1, \ldots, V_{\chi(G)}$ be the color classes. Each color class is an independent set, and by Lemma 4.1 the corresponding vectors can be packed on one machine with height at most 1. Since $\chi(G) \leq m$, the vectors corresponding to each color class can be packed in a separate machine. ∎

LEMMA 4.10. *If the schedule height for I is bounded by B, then $\chi(G) \leq \beta n^{1-1/2B} \ln n$ for some fixed constant β.*

Proof. Let $V_1, V_2, \ldots, V_m$ be the partition of vertices of G induced by the assignment of the vectors to the machines. Let G_i be the subgraph of G induced by the vertex set V_i. From Lemma 4.2 we have $\omega(G_i) \leq B$. Using Lemma 4.8 we obtain that $\chi(G_i) \leq \beta n^{1-1/B} \ln n$ for $1 \leq i \leq m$. Therefore it follows that $\chi(G) \leq \sum_i \chi(G_i) \leq m \cdot \beta n^{1-1/B} \ln n \leq \beta n^{1-1/2B} \ln n$. ∎

Proof of Theorem 4.7. Feige and Kilian [5] showed that unless $ZPP = NP$ for every $\epsilon > 0$ there is no polynomial time algorithm to approximate the chromatic number to within a factor of $n^{1-\epsilon}$. Suppose there is a B-approximation for the

VS problem. Lemmas 4.9 and 4.10 establish that if $\chi(G) \leq n^{1/2B}$, then we can infer by running the B-approximation algorithm for the VS problem that $\chi(G) \leq \beta n^{1-1/2B} \ln n$. This implies a $\beta n^{1-1/2B} \ln n$-approximation to the chromatic number. From the result of [5] it follows that this is not possible unless $NP = ZPP$. □

4.3. Vector bin packing. As mentioned before, VBP is APX-hard even for $d = 2$. We show a simple $d^{\frac{1}{2}-\epsilon}$ hardness for VBP when d is arbitrary. The reduction is similar to that in the previous subsection and uses hardness of graph coloring. Given a graph G with n vertices and m edges we create an instance I_G of VBP with n vectors, each of m dimensions. For a vector v_i corresponding to a vertex i of G, the jth coordinate is 1 if i is incident on the jth edge, otherwise it is 0. If the vectors are required to be packed into bins of height 1, it is easily seen that the number of bins required corresponds exactly to a coloring of G. Thus the hardness of chromatic number applies directly to VBP.

THEOREM 4.11. *Unless $NP = ZPP$, VBP is hard to approximate to within a $d^{\frac{1}{2}-\epsilon}$ factor for every fixed $\epsilon > 0$.*

5. Conclusions. We studied multidimensional generalizations of multiprocessor scheduling, bin packing, and the knapsack problem and obtained a variety of new algorithmic as well as inapproximability results for them. While our work gives new insights into the approximability of these problems, several questions remain open. In particular, large gaps remain between the upper and lower bounds for both VS and VBP when the number of dimensions is arbitrary. For packing integer programs, our hardness result is essentially tight, but it applies only to fixed values of B. The quality of approximation improves dramatically when B is allowed to grow as a logarithmic function of d; in particular a constant factor approximation is achievable when $B = \Omega(\log d / \log \log d)$. It will be interesting to see if our techniques can be extended to obtain a hardness result when B is allowed to be a function of d.

Acknowledgments. We are grateful to Aravind Srinivasan for guiding us through the literature on packing integer programs and his encouragement at the early stages of this research. We thank Cliff Stein for his comments on this work in its form as a chapter in the dissertation of the first author. We also thank the anonymous referees for many useful comments on an earlier version of the paper.

REFERENCES

[1] N. ALON, Y. AZAR, J. CSIRIK, L. EPSTEIN, S. V. SEVASTIANOV, A. P. A. VESTJENS, AND G. J. WOEGINGER, *On-line and off-line approximation algorithms for vector covering problems*, Algorithmica, 21 (1998), pp. 104–118.

[2] S. F. ASSMAN, *Problems in Discrete Applied Mathematics*, Ph.D. thesis, Mathematics Department, Massachusetts Institute of Technology, Cambridge, MA, 1983.

[3] C. CHEKURI AND S. KHANNA, *On multi-dimensional packing problems*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 185–194.

[4] D. J. DEWITT AND J. GRAY, *Parallel database systems: The future of high performance database systems*, Comm. ACM, 35 (1992), pp. 85–98.

[5] U. FEIGE AND J. KILIAN, *Zero knowledge and the chromatic number*, J. Comput. System Sci., 57 (1998), pp. 187–199.

[6] W. FERNANDEZ DE LA VEGA AND G. S. LUEKER, *Bin packing can be solved within $1 + \epsilon$ in linear time*, Combinatorica, 1 (1981), pp. 349–355.

[7] A. M. FRIEZE AND M. R. B. CLARKE, *Approximation algorithms for the m-dimensional* 0-1 *knapsack problem: Worst-case and probabilistic analyses*, European J. Oper. Res., 15 (1984), pp. 100–109.

[8] M. R. Garey and R. L. Graham, *Bounds for multiprocessor scheduling with resource constraints*, SIAM J. Comput., 4 (1975), pp. 187–200.
[9] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao, *Resource constrained scheduling as generalized bin packing*, J. Combin. Theory Ser. A, 21 (1976), pp. 257–298.
[10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.
[11] M. N. Garofalakis and Y. E. Ioannidis, *Scheduling issues in multimedia query optimization*, ACM Computing Surveys, 27 (1995), pp. 590–592.
[12] M. N. Garofalakis and Y. E. Ioannidis, *Multi-dimensional resource scheduling for parallel queries*, in Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, 1996, pp. 365–376.
[13] M. N. Garofalakis and Y. E. Ioannidis, *Parallel query scheduling and optimization with time-and space-shared resources*, in Proceedings of the 23rd International Conference on Very Large Data Bases, Morgan Kaufmann, San Francisco, 1997, pp. 296–305.
[14] M. N. Garofalakis, B. Özden, and A. Silberschatz, *Resource scheduling in enhanced pay-per-view continuous media databases*, in Proceedings of the 23rd International Conference on Very Large Data Bases, Morgan Kaufmann, San Francisco, 1997, pp. 516–525.
[15] R. L. Graham, *Bounds for certain multiprocessor anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1581.
[16] R. L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.
[17] M. M. Halldórsson, *Approximating k-set cover and complementary graph coloring*, in Proceedings of the Fifth IPCO Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 1084, Springer-Verlag, Berlin, 1996, pp. 118–131.
[18] J. Håstad, *Clique is hard to approximate to within* $n^{1-\epsilon}$, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, IEEE Comput. Soc. Press, Los Alamitos, CA, 1996, pp. 627–636.
[19] D. S. Hochbaum and D. B. Shmoys, *Using dual approximation algorithms for scheduling problems: Theoretical and practical results*, J. ACM, 34 (1987), pp. 144–162.
[20] O. H. Ibarra and C. E. Kim, *Fast approximation algorithms for the knapsack and sum of subset problems*, J. ACM, 22 (1975), pp. 463–468.
[21] N. Karmarkar and R. Karp, *An efficient approximation scheme for the one-dimensional bin-packing problem*, in Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, IEEE, New York, 1982, pp. 312–320.
[22] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela, *A probabilistic analysis of multi-dimensional bin packing problems*, in Proceedings of the Annual ACM Symposium on the Theory of Computing, 1984, pp. 289–298.
[23] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *Sequencing and scheduling: Algorithms and complexity*, in Handbooks in Operations Research and Management Science, Vol. 4, Elsevier Science, Amsterdam, 1993, pp. 445–522.
[24] L. Lovász, *On the ratio of the optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.
[25] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
[26] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer programs*, J. Comput. System Sci., 37 (1988), pp. 130–143.
[27] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.
[28] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley-Interscience Series in Discrete Mathematics, Wiley, Chichester, UK, 1986.
[29] A. Srinivasan, *Improved approximations of packing and covering problems*, in Proceedings of the 27th ACM Symposium on the Theory of Computing, 1995, pp. 268–276.
[30] G. Woeginger, *A polynomial time approximation scheme for maximizing the minimum completion time*, Oper. Res. Lett., 20 (1997), pp. 149–154.
[31] G. Woeginger, *There is no asymptotic PTAS for two-dimensional vector packing*, Inform. Process. Lett., 64 (1997), pp. 293–97.

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 852–869

COUNTING COMPLEXITY OF SOLVABLE BLACK-BOX GROUP PROBLEMS*

N. V. VINODCHANDRAN†

Abstract. We place many computational problems over solvable black-box groups in the counting complexity classes SPP or LWPP. The classes SPP and LWPP are considered classes of low counting complexity. In particular, SPP is *low* (powerless when used as oracles) for all gap-definable counting classes (PP, C$_=$P, Mod$_k$P, etc.) and LWPP is low for PP and C$_=$P. The results improve the upper bounds for these problems proved in [Arvind and Vinodchandran, *Theoret. Comput. Sci.*, 180 (1997), pp. 17–45], where the authors place these problems in randomized versions of SPP and LWPP. Because of the randomization, upper bounds in that paper implied lowness only for the class PP. The results in this paper favor the belief that these problems are unlikely to be complete for NP.

Key words. computational complexity theory, complexity classes, computational group theory

AMS subject classifications. 68Q15, 68Q17

DOI. 10.1137/S0097539703420651

1. Introduction.

Computational problems in group theory have recently been of considerable interest in computational complexity theory. One of the main reasons for this is that the exact complexity of many of the computational problems that arise from group theory are not exactly characterized. For example, consider the basic problem of testing membership in matrix groups over finite fields represented by a set of generators. No efficient algorithm is known for this problem even for the simplest case when the group involved in the problem instance is cyclic. While no polynomial time algorithm exists for many problems over matrix groups, it is not known whether it is hard for NP. The associated group-theoretic structure makes reductions to these problems difficult. Do these problems have complexity intermediate between P and NP? The results of this paper supports this belief. It is interesting to note that, while most of the natural problems that are not known to be in P are proved to be NP-complete, candidates for natural problems that are of intermediate complexity are very few. The well-known graph isomorphism problem, the problem of testing whether two labeled graphs are isomorphic, is one such candidate.

In order to study the complexity of computational group-theoretic problems in a generalized framework, Babai and Szemerédi [6] introduced the theory of *black-box* groups. Intuitively speaking, in this framework we have an infinite family of abstract groups. The elements of each group in the family are uniquely encoded as strings of uniform length. The group operations (product, inverse, etc.) are assumed to be provided by a group oracle and hence are easily computable. Black-box groups are subgroups of groups from such a family and are presented by generator sets. For example, matrix groups over finite fields and permutation groups presented by generator sets can be seen as examples of black-box groups. It is shown in [6] that

*Received by the editors January 2, 2003; accepted for publication (in revised form) February 15, 2004; published electronically May 5, 2004. Most of the research was done while the author was at the Institute of Mathematical Sciences, Chennai, India. Most of the results were presented in the 1997 FSTTCS conference [20].

http://www.siam.org/journals/sicomp/33-4/42065.html

†Department of Computer Science and Engineering, University of Nebraska, Lincoln, NE 68588 (vinod@cse.unl.edu).

many computational problems over black-box groups, including membership testing, are in NP.

A central problem considered in the papers of Babai and Szemerédi [6] and Babai [4] is order verification: given a black-box group G presented by a generator set and a positive integer n, verify that $|G| = n$. This problem is important because it turns out that several other problems reduce to order verification. In [4], using randomization to compute approximate lower bounds and sophisticated group-theoretic tools, it is shown that order verification for general black-box groups is in AM. As a consequence, it turns out that several problems for black-box groups are in NP $\cap$ co-AM. It follows that these problems cannot be NP-complete unless the polynomial hierarchy collapses to Σ_2^p [9, 19]. These results strongly indicate that these group-theoretic problems may be of intermediate complexity between P and NP.

In order to better understand the complexity status of these problems, researchers have taken alternative approaches. Investigating the *counting complexity* of these problems is one such approach. In these investigations, the classes of interest are counting complexity classes SPP and LWPP, defined and studied by Fenner, Fortnow, and Kurtz [12]. These classes can be seen as structural generalizations of the class UP. It holds that P $\subseteq$ UP $\subseteq$ SPP $\subseteq$ LWPP. The classes SPP and LWPP are of low counting complexity in the sense they do not add power to other counting classes such as PP and $C_=P$ when used as oracles. Membership of a problem in the class SPP or LWPP can be seen as evidence that the problem is unlikely to be hard for the class NP. First, intuitively we can say that problems that are in SPP or LWPP are of low counting complexity, and hence it is unlikely that these problems are NP-hard. Second, the classes SPP and LWPP are defined by imposing strong restrictions on the computation tree of nondeterministic Turing machines accepting languages in them. It would be surprising if all problems in NP can be accepted by Turing machines with such restrictions.

Köbler, Schöning, and Torán [16] studied the counting complexity of graph isomorphism and certain group-theoretic problems, such as group intersection, group factorization, etc., over permutation groups. They showed that these problems are in the counting class LWPP. Very recently, this upper bound has been improved to SPP by Arvind and Kurur [2].

The study of the counting complexity of black-box group problems was initiated by Arvind and Vinodchandran in [3]. There the authors study the counting complexity of a number of problems over a large subclass of groups called *solvable* black-box groups. Since solvable groups are a generalization of abelian groups, the authors first consider problems over *abelian* groups. Using a constructive version of a fundamental theorem on the structure of finite abelian groups, they show that, over abelian groups, the problems membership testing, order verification, group isomorphism, and group intersection are in the class SPP. They also show lowness of the problems group factorization, coset intersection, and double-coset membership over abelian groups for PP and $C_=P$.

In the case of solvable groups, using a randomized algorithm for computing the commutator series of a solvable black-box group from [5] and a constructive version of the fundamental theorem for abelian *factor* groups, in order to construct a special set of generators called the *canonical generator set*, the authors of [3] show that all the above-mentioned problems are in *randomized versions* of SPP or LWPP. While these randomized versions of SPP and LWPP are low for the class PP, the lowness properties of these problems for classes such as $C_=P$, $\mathrm{Mod}_k P$, and other gap-definable

counting classes were not clear from [3].

The present paper overcomes this randomization bottleneck. We show an upper bound of SPP (and in some cases LWPP) for a whole host of problems over solvable black-box groups. The main technical contribution of this paper is the design of a *deterministic* oracle Turing machine for converting a set of generators of a solvable group to a canonical generator set. This oracle algorithm can then be combined with the constructions in [3] to obtain the upper bounds for various problems. In particular, we show that, over solvable black-box groups, the problems solvability testing, membership testing, subgroup testing, normality testing, order verification, nilpotence testing, group isomorphism, and group intersection are in the class SPP. We also show that the problems group intersection, group factorization, coset intersection, and double-coset membership over solvable groups are in the class LWPP. Thus, at least with respect to their counting complexity, problems over solvable groups may not be harder than their counterparts over abelian groups.

The rest of the paper is organized as follows. In section 2, we give complexity-theoretic and group-theoretic notation and definitions which are necessary for the paper. The definition of a canonical generator set for solvable groups and results relating to it are also given. Section 3 is devoted to the design of a deterministic oracle algorithm, CanonicalGenerator, for computing a canonical generator set for a solvable group from an arbitrary generator set. In section 4, we give definitions of the computational problems that we are interested in, and we improve the upper bounds on the counting complexity of these problems using the algorithm given in section 3. Finally, in section 5, we conclude the paper with some open problems.

2. Preliminaries. In this section, we give relevant definitions from complexity theory and group theory. We also present complexity-theoretic as well as group-theoretic techniques that we use in this paper.

2.1. Complexity theory. We refer the reader to [7, 8] for standard complexity-theoretic definitions. Here we give only the minimal notation and definitions. We fix the finite alphabet $\Sigma = \{0, 1\}$. Let $A, B \subseteq \Sigma^*$ be two languages. The language $0A \cup 1B$ is denoted by $A \oplus B$. In [12], a uniform method for defining a number of counting classes using GapP, the class of gap-definable functions, is given. The classes that can be defined in this manner are called *gap-definable* classes. Refer to [12] for details. We give explicit definitions of only two classes, SPP and LWPP. Let GapP [12] denote the class of gap-definable functions. A language L is in LWPP if there are functions $f \in \text{GapP}$ and $h \in \text{FP}$ (h is positive) such that $x \in L$ implies that $f(x) = h(|x|)$, and $x \notin L$ implies that $f(x) = 0$. A language L is in SPP if there is an $f \in \text{GapP}$ such that $x \in L$ implies that $f(x) = 1$, and $x \notin L$ implies that $f(x) = 0$. It follows that $\text{SPP} \subseteq \text{LWPP}$.

The concept of lowness is a well-studied notion in structural complexity theory. Intuitively, we say that the class $\mathcal{L}$ is low for class $\mathcal{C}$ if $\mathcal{L}$ is powerless as an oracle to a machine accepting languages in $\mathcal{C}$. Next we give the formal definition of lowness.

Definition 2.1. *Let $\mathcal{C}$ be any complexity class which allows natural relativization. Then the class $\mathcal{L}$ is said to be* low *for $\mathcal{C}$ if for all $L \in \mathcal{L}$, $\mathcal{C}^L = \mathcal{C}$, where $\mathcal{C}^L$ denotes the complexity class that is obtained by relativizing $\mathcal{C}$ with respect to the language L.*

The main interest in these classes is because of the following theorem proved in [12] regarding their complexity. This result indicates that SPP and LWPP are counting classes of low complexity. It is believed that those problems in SPP or LWPP cannot be complete for NP.

THEOREM 2.2 (see [12]). *The class* SPP *is low for all gap-definable counting classes. The class* LWPP *is low for* PP *and* $C_=P$.

Let M be an oracle NP machine, let $A \in$ NP be accepted by an NP machine N, and let $f \in$ FP. We say that M makes $f(n)$-*guarded* queries to A if, on length n inputs, M asks only queries y for which $N(y)$ has either 0 or $f(n)$ accepting paths for each n. In this terminology, we state a weaker version of a theorem from [16] which gives a method for proving membership of languages in the classes SPP and LWPP.

THEOREM 2.3 (see [16]). *Let M be a deterministic polynomial time oracle machine and let f be a polynomial time computable function. If $A \in$ NP such that M makes $f(n)$-guarded queries to A, then there is a polynomial q such that the function h, where $h(x) = f(|x|)^{q(|x|)}$ if M on input x accepts and $h(x) = 0$ if M on input x rejects, is in* GapP.

For proving our upper bounds we use the following corollary of the above theorem.

COROLLARY 2.4. *Let L be a language accepted by a deterministic polynomial time oracle machine M with queries to a language A in* NP. *Let f be a polynomial time computable function.*

1. *If M makes* 1*-guarded queries to A, then $L \in$* SPP.
2. *If M makes $f(n)$-guarded queries to A, then $L \in$* LWPP.

2.2. Group theory. Here we give some notation and basic definitions from group theory. We also state some basic results. For further results and their proofs, please refer to standard textbooks [10, 13].

Let G be a group. A subset H of G is called a *subgroup* of G (denoted $H < G$ or $G > H$) if H is a group under the group operation of G. For a subset S of G, the smallest subgroup of G containing S is called the group *generated by* S and is denoted by $\langle S \rangle$. This group is the same as the set of all finite products of elements from S. A subset S of G is a generator set for G if G is identical to $\langle S \rangle$. A group G is *finite* if the cardinality of the set G is finite. In this paper, we are only interested in finite groups. Henceforth by a group we mean a finite group. The *order* of G is defined as the cardinality of the set G and is denoted by $|G|$. A group is said to be cyclic if it is generated by a single element. For an element $g \in G$, the *order* of g (denoted as $o(g)$) is the order of the cyclic subgroup generated by g. This is the same as the smallest positive integer k such that $g^k = e$ (g^k denotes the product of g, k times), where e is the identity of G. Two groups G and H are said to be isomorphic if there is a bijection (set-theoretic) ϕ from G to H such that for any $x, y \in G$, $\phi(xy) = \phi(x)\phi(y)$. Isomorphism preserves all structural properties of groups.

A fundamental theorem in finite group theory, due to Lagrange, states that if $H < G$, then $|H|$ divides $|G|$. This theorem has a large number of algorithmic applications. For example, it follows from Lagrange's theorem that any finite group G is generated by a set of group elements of cardinality bounded by $\log |G|$.

Let H be a subgroup of a group G. For $g \in G$ the set $\{hg \mid h \in H\}$, denoted by Hg, is called a *right coset* of H in G. Similarly, the set $gH = \{gh \mid h \in H\}$ is called a *left coset* of H in G. H is a *normal* subgroup of G if for all $g \in G$ it holds that $Hg = gH$. A fundamental result in the theory of groups is that if H is a normal subgroup of G, then the set of right cosets of H in G forms a group (called the *factor group* or a *quotient group* induced by H and denoted by G/H) under the binary operation $\cdot$ defined as $Hx \cdot Hy = Hxy$. The identity element of this group is the coset $He = H$. For a set $X \subseteq G$, the *normal closure* of X is the smallest normal subgroup containing X.

Solvable groups. Here we give the definition of a *solvable* group and state some properties of them. Intuitively solvable groups can be thought of as a generalization of *abelian groups*. A group G is *abelian* if for all $x, y \in G$, $xy = yx$; that is, $xyx^{-1}y^{-1} = e$. In general, the element $xyx^{-1}y^{-1}$ is called the *commutator* of elements x and y in G. The subgroup of G generated by the set $\{xyx^{-1}y^{-1} \mid x, y \in G\}$ is called the *commutator subgroup* of G. We denote this subgroup by G'. Observe that if G is abelian, G' is the trivial group containing only the identity element. The commutator subgroup G' is actually a normal subgroup of G and the factor group G/G' is abelian. For a group G, the sequence $G = G_0 > G_1 > \cdots$ is called the commutator sequence, where each group G_i is the commutator subgroup of G_{i-1}. G is solvable if the commutator sequence terminates in the trivial subgroup $\langle e \rangle$ in finitely many steps. This intuitively means that any solvable group can be decomposed into a series of abelian factor groups. Therefore understanding the structure of a solvable group boils down to understanding the structure of the abelian factor groups involved. Next we state a structure theorem for abelian groups which essentially states that any abelian group can be uniquely decomposed into a set of cyclic groups. We need some more definitions to state the theorem.

Let p be a prime. A p-group is a finite group whose order is a power of p. Let G be finite group such that $|G| = p_1^{e_1} p_2^{e_2} \dots p_r^{e_r}$. The existence of subgroups in G which are p-groups is given by Sylow's theorem. That is, for each i there is a subgroup of G of order $p_i^{e_i}$. A subgroup of G of order $p_i^{e_i}$ is referred to as a *p_i-Sylow subgroup* of G.

THEOREM 2.5 (see [10]). *Let G be a finite abelian group such that $|G| = p_1^{e_1} p_2^{e_2} \dots p_r^{e_r}$, where the p_i's are distinct primes. The group G can be expressed as the direct product of its Sylow subgroups $S(p_1), S(p_2), \dots, S(p_r)$, where $|S(p_i)| = p_i^{e_i}$ for $1 \le i \le r$. Furthermore, for $1 \le i \le r$, each Sylow subgroup $S(p_i)$ can be uniquely expressed as the direct product of cyclic groups of orders $p_i^{e_{i1}}, p_i^{e_{i2}}, \dots, p_i^{e_{is_i}}$ such that $e_{i1} \ge e_{i2} \ge \cdots \ge e_{is_i}$ and $\Sigma_{j=1}^{s_i} e_{ij} = e_i$. This decomposition of G is unique up to isomorphism.*

Solvable groups form a large subclass of all finite groups. In fact, a celebrated result due to Fiet and Thompson says that any finite group of odd order is solvable. Subgroups of solvable groups are solvable. It follows from Lagrange's theorem that, if G is solvable, then the length of the commutator sequence is bounded by $\log |G|$. From a computational viewpoint this fact is very useful. It also holds that two solvable groups G and H are isomorphic if and only if the factor group H_{i-1}/H_i is isomorphic to G_{i-1}/G_i for all i where H_i (G_i) is the ith element in the commutator series of H (respectively, G).

We will also consider nilpotent groups. Let H_1, H_2 be two subgroups of G. The *mutual commutator subgroup* of H_1 and H_2 is the group generated by the set $\{xyx^{-1}y^{-1} \mid x \in H_1,\ y \in H_2\}$ and is denoted by $[H_1, H_2]$. Note that using this notation the commutator subgroup G' of G is $[G, G]$. The chain of subgroups $G = L^0(G) > L^1(G) > \cdots$, where $L^i(G) = [G, L^{i-1}(G)]$, is called the *lower central series* of G. G is called *nilpotent* if the series terminates in the trivial subgroup $\{e\}$. A nilpotent group is also solvable.

Computational problems over black-box groups. Now we define the notion of black-box groups.

DEFINITION 2.6. *A* group family *is a countable sequence $\mathcal{B} = \{B_m\}_{m \ge 1}$ of finite groups B_m, such that there are polynomials p and q satisfying the following conditions. For each $m \ge 1$, elements of B_m are uniquely encoded as strings in $\Sigma^{p(m)}$. The group operations (inverse, product, and testing for identity) of B_m can be performed in time*

bounded by $q(m)$ for every $m \geq 1$. The order of B_m is computable in time bounded by $q(m)$ for each m. We refer to the groups B_m of a group family and their subgroups (presented by generators sets) as black-box *groups. A class $\mathcal{C}$ of finite groups is said to be a* subclass *of $\mathcal{B}$ if every $G \in \mathcal{C}$ is a subgroup of some $B_m \in \mathcal{B}$.*

Remark. The black-box groups we consider are a somewhat restricted version of the black-box groups introduced in [6]. The main difference is that, while we require identity testing to be in polynomial time, they require only that it be in *nondeterministic* polynomial time. This naturally places factor groups within the framework of black-box groups. Since in [6] the authors focus on proving many group-theoretic problems in NP, this generality does not cause additional complications. Since we are interested on certain "uniqueness" properties, we need this restriction.

Let S_n denote the permutation group on n elements. Then $\{S_n\}_{n\geq 1}$ is a group family of all permutation groups S_n. As another example let $GL_n(q)$ denote the group of all $n \times n$ invertible matrices over the finite field F_q of size q. The collection $\mathrm{GL(q)} = \{GL_n(q)\}_{n\geq 1}$ is a group family. The class of all solvable subgroups, $\{G \mid G < GL_n(q)$ for some n and G is solvable$\}$ is a subclass of GL(q).

The following decision problems which we consider in this paper are well studied in computational group theory [4, 6, 11, 16]. Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family.

Solvability testing $\stackrel{\triangle}{=} \{(0^m, S) \mid \langle S\rangle < B_m$ and $\langle S\rangle$ is solvable$\}$.
Membership testing $\stackrel{\triangle}{=} \{(0^m, S, g) \mid \langle S\rangle < B_m$ and $g \in \langle S\rangle\}$.
Subgroup testing $\stackrel{\triangle}{=} \{(0^m, S_1, S_2) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $\langle S_1\rangle$ is a subgroup of $\langle S_2\rangle\}$.
Normality testing $\stackrel{\triangle}{=} \{(0^m, S_1, S_2) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $\langle S_1\rangle$ is a normal subgroup of $\langle S_2\rangle\}$.
Nilpotence testing $\stackrel{\triangle}{=} \{(0^m, S) \mid \langle S\rangle < B_m$ and $\langle S\rangle$ is nilpotent$\}$.
Order verification $\stackrel{\triangle}{=} \{(0^m, S, n) \mid \langle S\rangle < B_m$ and $|\langle S\rangle| = n\}$.
Group isomorphism $\stackrel{\triangle}{=} \{(0^m, S_1, S_2) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and are isomorphic$\}$.
Group intersection $\stackrel{\triangle}{=} \{(0^m, S_1, S_2) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $\langle S_1\rangle \cap \langle S_2\rangle \neq (e)\}$.
Group factorization $\stackrel{\triangle}{=} \{(0^m, S_1, S_2, g) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $g \in \langle S_1\rangle\langle S_2\rangle\}$.
Coset intersection $\stackrel{\triangle}{=} \{(0^m, S_1, S_2, g) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $\langle S_1\rangle g \cap \langle S_2\rangle \neq \emptyset\}$.
Double coset memb $\stackrel{\triangle}{=} \{(0^m, S_1, S_2, g, h) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $g \in \langle S_1\rangle h\langle S_2\rangle\}$.

The problems group factorization, coset intersection, and double coset memb are very closely related. In particular, over permutation groups these problems are Turing equivalent to each other [15, 18]. Also the graph isomorphism problem is a special case of double coset memb over permutation groups [14].

Group-theoretic techniques. We are interested in the counting complexity of the problems when the groups involved are solvable. Since solvable groups are a generalization of abelian groups, some remarks about the complexity of these problems over abelian black-box groups are in order. For proving tight upper bounds on the counting complexity of the above-mentioned problems over abelian groups in [3], the authors employ a constructive version of the structure theorem (Theorem 2.5). One of the immediate consequences of this theorem is the existence of a special generator set, called the *independent generator set*, for any abelian group. To be precise, let G be a finite abelian group. An element $g \in G$ is said to be *independent* of a set $X \subseteq G$

if $\langle g\rangle \cap \langle X\rangle = \{e\}$. A generator set S of G is an *independent generator set* for G if all $g \in S$ is independent of $S - \{g\}$. A set of generators of the cyclic groups involved forms an independent generator set for any abelian group. One of the very useful properties of independent generator sets is the following. Let S be an independent generator set for an abelian group G. Then for any $g \in G$, there exist *unique* indices l_h for $h \in S$; $l_h < o(h)$ such that $g = \prod_{h\in S} h^{l_h}$. Hence membership testing in G can be done in a 1-guarded manner if G is presented by an independent generator set. In [3], an algorithm for converting a given generator set to an independent generator set is given, which is used in proving the upper bounds on the counting complexity for problems over abelian black-box groups.

For proving the upper bounds for problems over solvable black-box groups in [3], the authors introduce a generalization of the notion of independent generator set, called the *canonical generator set* for any class of finite groups. We now give the definition of a canonical generator set. The existence of canonical generators for the class of solvable groups is shown in [3].

DEFINITION 2.7. *Let $\mathcal{B} = \{B_m\}_{m>0}$ be any group family. Let $\mathcal{C}$ be a subclass of $\mathcal{B}$. The class of groups $\mathcal{C}$ has* canonical generator sets *if for every $G \in \mathcal{C}$, if $G < B_m$, there is an* ordered *set $S = \{g_1, g_2, \ldots, g_s\} \subseteq G$ such that each $g \in G$ can be* uniquely *expressed as $g = g_1^{l_1} g_2^{l_2} \ldots g_s^{l_s}$, where $0 \le l_i < o(g_i)$, $1 \le i \le s$. Furthermore, $s \le q(m)$ for a polynomial q. S is called a* canonical generator set *for G.*

Notice that the above definition is a generalization of the definition of an independent generator set in the sense that the uniqueness property of the indices is preserved. Define a language L as follows:

$$L = \left\{ (0^m, S, g) | S \subseteq B_m, g \in B_m; \forall h \in S\ \exists l_h; 0 \le l_h < o(h) \text{ and } g = \prod_{h\in S} h^{l_h} \right\}.$$

The following proposition brings out the fact that the language L can act as a "pseudo membership testing" in the sense that if S is a canonical generator set, then $(0^m, S, g) \in L$ if and only if $g \in \langle S\rangle$. More importantly in this case, the NP machine M_u (given in the proposition) will have a *unique* accepting path for those instances inside L.

PROPOSITION 2.8. *Let $\mathcal{B} = \{B_m\}_{m>0}$ be any group family. Then there exists an* NP *machine M_u witnessing $L \in$ NP with the following property. Let $\mathcal{C}$ be a subclass of $\mathcal{B}$ which has a canonical generator set, and suppose the input to M_u satisfies the promise that S is a canonical generator set for $\langle S\rangle \in \mathcal{C}$. Then M_u on input $(0^m, S, g)$ will have a unique accepting path if $g \in \langle S\rangle$, and M_u will have no accepting path if $g \notin \langle S\rangle$. The behavior of M_u is unspecified if the input does not satisfy the promise.*

Proof. We know that checking whether a number is prime or not is in P [1]. Using this, one can easily design an unambiguous nondeterministic polynomial time transducer which computes the prime factorization of any number. Let M' be such a machine. Now, it is easy to see that the order of any $g \in B_m$ can be computed if the prime factorization of $|B_m|$ is given. So, M_u first computes $|B_m|$ in polynomial time. Then by simulating M', it computes the prime factorization of $|B_m|$ and computes the order $o(h)$ for all $h \in S$. Now, M_u guesses indices l_h such that $1 \le l_h < o(h)$ and accepts if $g = \prod_{h\in S} h^{l_h}$ and rejects otherwise. From the definition of canonical generator sets, it follows that M_u has the behavior as described in the proposition. ∎

The next lemma shows the existence of canonical generator sets for any solvable group.

LEMMA 2.9 (see [3, Lemma 3.4]). *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family such that $|B_m| \leq 2^{q(m)}$ for a polynomial q. Let $G < B_m$ be a finite solvable group and let $G = G_0 > G_1 > \cdots > G_{k-1} > G_k = e$ be the commutator series of G. Let $T_i = \{h_{i1}, h_{i2}, \ldots, h_{is_i}\}$ be a set of distinct coset representatives corresponding to an independent set of generators for the abelian group $H_i = G_{i-1}/G_i$. Then for any i, $1 \leq i \leq k$, the ordered set*[1] *$S_i = \bigcup_{j=i}^{k} T_j$ forms a canonical generator set for the group G_i and $|S_i| \leq q(m)$. Thus the class of solvable groups from $\mathcal{B}$ has canonical generator sets.*

The basic steps implicitly involved in the upper bound proofs given in [3], for problems over solvable black-box groups, are the following:

1. A deterministic oracle algorithm (let us call it CANONIZE) is developed which takes an arbitrary set of generators for the commutator series of a solvable black-box group as input and converts it into a canonical generator set by making 1-guarded queries to a language in NP.
2. By carefully combining the algorithm CANONIZE with a randomized algorithm from [5] for computing generator sets of the commutator series for any solvable black-box group, a randomized oracle algorithm (let us call it RANDCANONICALGENERATOR) for converting a generator set for any solvable group to a canonical generator set (which makes 1-guarded queries to an NP language) is given.
3. RANDCANONICALGENERATOR is then easily modified to give membership of many computational problems over solvable groups in randomized counting classes which are low for PP.

In this paper, we avoid the randomization involved in step 2. More precisely, by using the algorithm CANONIZE as a subroutine, we give a *deterministic* oracle algorithm CANONICALGENERATOR (which makes 1-guarded queries to an NP language) for converting an arbitrary generator set to a canonical generator set for any solvable black-box group G. This will immediately give improved upper bounds on the counting complexity of many problems over solvable groups. In the next section we present the algorithm CANONICALGENERATOR for converting an arbitrary generator set to a canonical generator set for any solvable group.

Since we will be using the algorithm CANONIZE as a subroutine in CANONICALGENERATOR, we describe the behavior of CANONIZE as a theorem. First we state a result (Corollary 4.9 from [3]) regarding abelian factor groups which is the building-block for CANONIZE.

LEMMA 2.10 (see [3, Corollary 4.9]). *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. There is a deterministic oracle machine M_f that takes as input tuples $(0^m, X, Y)$, where $X, Y \subseteq B_m$ are finite sets, and a language $L_f \in$ NP as oracle. Let $G = \langle X \rangle$ and $H = \langle Y \rangle$ and suppose the input $(0^m, X, Y)$ satisfies the following properties:*

1. *Y is a canonical generator set for H.*
2. *H is a normal subgroup of G and the factor group G/H is abelian.*

Then the machine outputs a list of coset representatives corresponding to an independent generator set for G/H and also outputs $|G/H|$. Furthermore, it runs in time polynomial in input length, and it makes 1-guarded queries to the NP *oracle L_f. The behavior of the machine is not specified if the input does not satisfy the properties.*

The following theorem describes the behavior CANONIZE. The proof essentially follows from an iterative application of the above lemma. We omit the details.

[1] The elements of the set $\bigcup_{j=i}^{k} T_j$ are ordered on increasing values of the index j, and lexicographically within each set T_j.

THEOREM 2.11. *Let $\mathcal{B} = \{B_m\}_{m\geq 0}$ be a group family. Then there is a deterministic oracle machine* CANONIZE *and a language $L' \in$ NP such that* CANONIZE *takes $\langle 0^m, S_0, \ldots, S_k\rangle$, $S_i \subseteq B_m$ as input and L' as oracle. Suppose the input satisfies the promise that $\langle S_0\rangle$ is solvable and for $0 \leq i \leq k$, S_i generates the ith commutator subgroup of $\langle S_0\rangle$. Then* CANONIZE *outputs canonical generator sets for $\langle S_i\rangle$ for $0 \leq i \leq k$. Moreover,* CANONIZE *runs in time polynomial in the length of the input and makes only 1-guarded queries to L'. The behavior of* CANONIZE *is unspecified if the input does not satisfy the promise.*

3. Computing a canonical generator set. This section is devoted to the proof of the following theorem.

THEOREM 3.1. *Let $\mathcal{B} = \{B_m\}_{m\geq 0}$ be a group family. Then there is a language $L_{ca} \in$ NP and a deterministic oracle machine* CANONICALGENERATOR *that takes $(0^m, S)$ as input and L_{ca} as oracle, and outputs canonical generator set for all the groups in the commutator series of $\langle S\rangle$ if $\langle S\rangle$ is solvable and outputs* NOT SOLVABLE *otherwise. Moreover,* CANONICALGENERATOR *runs in time polynomial in the length of the input and makes only 1-guarded queries to L_{ca}.*

Before going into the formal proof of the theorem, we give basic ideas behind the proof. Let S be a set of generators for a solvable group. Let $\langle S\rangle = G_0 > \cdots > G_i > \cdots > G_k = \{e\}$ be the commutator series of $\langle S\rangle$. We are interested in computing generator sets for all G_i. The following theorem based on normal closure provides a method for computing a generator set for the commutator subgroup of any group. Recall that the normal closure of $X \subseteq G$ is the smallest normal subgroup of G which contains X.

THEOREM 3.2. *Let G be a finite group generated by the set S. Then the commutator subgroup of G is the normal closure of the set $\{ghg^{-1}h^{-1} \mid g, h \in S\}$ in G.*

Proof. The proof uses standard group-theoretic argument. Let N be the normal closure of the set $X = \{ghg^{-1}h^{-1} \mid g, h \in S\}$ in G and let G' be the commutator subgroup of G. Recall that $G' = \langle\{ghg^{-1}h^{-1} \mid g, h \in G\}\rangle$. It is a well-known group-theoretic fact that G' is normal in G and G/G' is abelian. Moreover if H is a normal subgroup of G such that G/H is abelian, then $G' < H$. Now, from the definition of N and G' and the fact that G' is normal in G, it follows that $N < G'$. Hence to show that $N = G'$, it is enough to show that G/N is abelian. First, observe that G/N is generated by $\{gN \mid g \in S\}$. But for all $g, h \in S$, $(gN)(hN)(gN)^{-1}(hN)^{-1} = N$. Hence the theorem. □

The above theorem gives us the following easy polynomial time oracle algorithm COMMUTATORSUBGROUP, which takes $(0^m, S)$ as input and membership testing as oracle and computes a generator set for the commutator group of $\langle S\rangle$.

COMMUTATORSUBGROUP$(0^m, S)$
1 $X \leftarrow \{ghg^{-1}h^{-1} \mid g, h \in S\}$
2 **while** $\exists g \in S$; $x \in X$ such that $(0^m, X, gxg^{-1}) \notin$ membership testing
3 **do** $X \leftarrow X \cup gxg^{-1}$
4 **end-while**
5 **Output** X

CLAIM 3.2.1. *On input $(0^m, S)$,* COMMUTATORSUBGROUP *runs in polynomial time and outputs a generator set for the commutator subgroup of $\langle S\rangle$.*

Proof. We can argue that COMMUTATORSUBGROUP outputs a generator set for the commutator subgroup of $\langle S\rangle$ as follows. From Theorem 3.2, it follows that we need a generator set for the smallest normal subgroup of $\langle S\rangle$ containing X. Moreover

it easily follows from the definition that, in order to check whether $H = \langle X \rangle$ is normal in $G = \langle S \rangle$, it is enough to check for all pairs $g \in S$ and $x \in X$ whether $gxg^{-1} \in H$. The algorithm adds new elements of the form gxg^{-1} to X until this condition is satisfied. Therefore the output of this algorithm generates a normal subgroup of $\langle S \rangle$ containing X. Since all the elements added should be contained in any normal subgroup of $\langle S \rangle$ containing X, the output generates the smallest such normal subgroup. To argue that CommutatorSubgroup runs in polynomial time, let X_i be the set X at the beginning of the ith iteration of the **while**-loop. If, after the ith iteration, no new element is added to X_i, then X_i is output. Otherwise, if $X_{i+1} = X_i \cup \{g\}$, it follows from Lagrange's theorem that $|\langle X_{i+1} \rangle| \geq 2|\langle X_i \rangle|$. Hence the number of iterations of the **while**-loop is bounded by the polynomial $p(m)$. □

Since in the above algorithm, the queries to the membership testing oracle may not be 1-guarded, a straightforward adaptation of the algorithm for computing a generator set for all elements in the commutator series seems difficult. Suppose we can make sure that whenever a query $y = \langle 0^m, X, g \rangle$ to membership testing is made, X is a canonical generator set for the solvable group $\langle X \rangle$. Then from Proposition 2.8 it follows that we can replace the membership testing oracle with the NP language L and the query y will be 1-guarded. We ensure this promise by constructing the commutator series in stages.

Let S_l^j denote the partial generator set for the lth element in the commutator series of G_0 constructed at the beginning of *stage* j. At *stage* 1 we have $S_0^1 = S$ and $S_i^1 = \{e\}$ for $1 \leq l \leq p(m)$, where p is the polynomial bounding the length of any element in the group family. Input to *Stage* j is the tuple $\langle i, S_i^j, \ldots, S_{p(m)}^j \rangle$ such that for $k > i$, S_k^j is a canonical generator set for the solvable group $\langle S_k^j \rangle$. At this stage, the only oracle queries made are with generator sets S_k^j for $k > i$ and, in particular, will not involve S_i^j. This allows us to make only queries that are 1-guarded. At the end of the stage, we update each S_l^j to S_l^{j+1} such that S_l^{j+1} is still a subgroup of G_l, the lth commutator subgroup of G_0. To keep the running time within polynomial bound, we make sure that after $p(m)$ stages, there exists k, such that the kth partial commutator subgroup doubles in size. Then from Lagrange's theorem, it will follow that the commutator series will be generated after $p^3(m)$ stages. We now formally prove the theorem.

Proof of Theorem 3.1. We will first give the formal description of the algorithm CanonicalGenerator and then prove the correctness.

CanonicalGenerator uses oracle algorithms CheckCommutator and Canonize as subroutines. CheckCommutator takes as input $(0^m, X, Y)$ such that $X, Y \subseteq B_m$ and checks whether $\langle Y \rangle$ contains the commutator subgroup of $\langle X \rangle$. This is done by first checking whether the commutators of all the elements in X are in $\langle Y \rangle$. If this is not the case, the algorithm returns a commutator not in $\langle Y \rangle$. Otherwise, it further checks whether $\langle Y \rangle$ is normal in $\langle X \rangle$. Notice that to do this it is enough to verify that for all $x \in X$ and $y \in Y$, $xyx^{-1} \in \langle Y \rangle$. If this condition is false, the algorithm returns an element $xyx^{-1} \notin \langle Y \rangle$. If both the conditions are true, it follows from Theorem 3.2 that $\langle Y \rangle$ contains the commutator subgroup of $\langle X \rangle$.

CheckCommutator makes oracle queries to the language L (defined in the previous section) for testing membership in $\langle Y \rangle$. It should be noted that, for CheckCommutator to work as intended, Y should be a canonical generator set for the group $\langle Y \rangle$. Also the definition of normality requires $\langle Y \rangle < \langle X \rangle$. We will make sure that CanonicalGenerator makes calls to CheckCommutator with $(0^m, X, Y)$ as input only when Y is a canonical generator set for the solvable group $\langle Y \rangle$ and

$\langle Y \rangle < \langle X \rangle$. A formal description of the subroutine CheckCommutator is given below.

CheckCommutator$(0^m, X, Y)$
if $\exists x_1, x_2 \in X$, such that $(0^m, Y, x_1 x_2 x_1^{-1} x_2^{-1}) \notin L$
  then $g \leftarrow x_1 x_2 x_1^{-1} x_2^{-1}$
    Return g
  else **if** $\exists x \in X, y \in Y$ such that $(0^m, Y, xyx^{-1}) \notin L$
    then $g \leftarrow xyx^{-1}$
      Return g
    else $g \leftarrow$ YES
      Return g
    end-if
end-if

The subroutine Canonize is the algorithm promised by Theorem 2.11 for computing a canonical generator set for a solvable black-box group G, given an arbitrary generator set for the commutator series of G. Canonize makes 1-guarded queries to the NP language L' if the input satisfies the promise given in Theorem 2.11. For $0 \leq k \leq l \leq p(m)$, let $[Canonize(S_k^j, \ldots, S_{p(m)}^j)]_l$ denote the generator set produced by Canonize for the group $\langle S_l^j \rangle$.

The following is the description of the algorithm CanonicalGenerator. Define the language L_{ca} as $L_{ca} = L' \oplus L$. Notice that the oracle access to L_{ca} is implicit in the description. That is, CanonicalGenerator queries L' through the subroutine Canonize and L through CheckCommutator.

CanonicalGenerator$(0^m, S)$
$S_0^1 \leftarrow S$; $S_i^1 \leftarrow \{e\}$ for $1 \leq i \leq p(m)$
$i \leftarrow 0$
$j \leftarrow 1$
Stage j (Input to this stage is $\langle i, S_i^j, \ldots, S_{p(m)}^j \rangle$)
$k \leftarrow i$
$g \leftarrow CheckCommutator(0^m, S_k^j, S_{k+1}^j)$
while $g \neq$ YES
  do $S_{k+1}^j \leftarrow S_{k+1}^j \cup \{g\}$
    $k \leftarrow k + 1$
    if $k = p(m)$
      then Output NOT SOLVABLE
    end-if
    $g \leftarrow CheckCommutator(0^m, S_k^j, S_{k+1}^j)$
end-while
if $k = 0$
  then Output $[Canonize(S_0^j, S_1^j, \ldots, S_{p(m)}^j)]_l$ for all $0 \leq l \leq p(m)$
  else $S_l^{j+1} \leftarrow S_l^j$ for $1 \leq l \leq (k-1)$
    $S_l^{j+1} \leftarrow [Canonize(S_k^j, S_{k+1}^j, \ldots, S_{p(m)}^j)]_l$ for $k \leq l \leq p(m)$
    $i \leftarrow (k-1)$
    goto **Stage** $j+1$
end-if

Now we are ready to prove the correctness of CanonicalGenerator. We first prove a series of claims, from which the correctness will follow easily.

CLAIM 3.2.2. *In the algorithm* CANONICALGENERATOR*, at any stage j, it holds that for all i, $1 \leq i < p(m)$, $\langle S_{i+1}^j \rangle < \langle S_i^j \rangle'$.*

Proof. We prove this by induction on the stages. For the base case, when $j = 0$, it is clear that the claim holds. Assume that it is true for the $(j-1)$th stage. Now consider S_{i+1}^j and S_i^j. Depending on how the sets S_{i+1}^j and S_i^j are updated in lines 17 and 18 of CANONICALGENERATOR, we have the following cases.

Case 1. $S_i^j = S_i^{j-1}$; $S_{i+1}^j = S_{i+1}^{j-1}$. In this case, from the induction hypothesis, it is clear that $\langle S_{i+1}^j \rangle < \langle S_i^j \rangle'$.

Case 2. $S_i^j = S_i^{j-1} \cup \{g_i\}$; $S_{i+1}^j = S_{i+1}^{j-1}$. From the induction hypothesis, it follows that $\langle S_{i+1}^j \rangle = \langle S_{i+1}^{j-1} \rangle < \langle S_i^{j-1} \rangle' < \langle S_i^j \rangle'$.

Case 3. $S_i^j = S_i^{j-1}$; $S_{i+1}^j = S_{i+1}^{j-1} \cup \{g_{i+1}\}$. The element g_{i+1} is added to the set S_{i+1}^{j-1} at line 8 of the algorithm, where g_{i+1} is the element returned by the subroutine CHECKCOMMUTATOR. Suppose g_{i+1} is a commutator of the set $S_i^j = S_i^{j-1}$. Then $g_{i+1} = xyx^{-1}y^{-1}$ for some elements $x, y \in S_i^j$. From induction hypothesis we have that $\langle S_{i+1}^{j-1} \rangle < \langle S_i^{j-1} \rangle' = \langle S_i^j \rangle'$. Also $g_{i+1} \in \langle S_i^j \rangle'$ since g_{i+1} is a commutator of S_i^j. Therefore $\langle S_{i+1}^j \rangle = \langle S_{i+1}^{j-1} \cup \{g_{i+1}\} \rangle < \langle S_i^j \rangle'$. On the other hand, suppose g_{i+1} is of the form xyx^{-1} for some $x \in S_i^j = S_i^{j-1}$ and $y \in S_{i+1}^{j-1}$. We have $\langle S_{i+1}^{j-1} \rangle < \langle S_i^{j-1} \rangle' = \langle S_i^j \rangle'$. But we know that $\langle S_i^j \rangle'$ is normal in $\langle S_i^j \rangle$. So, in particular $g_{i+1} \in \langle S_i^j \rangle'$. Therefore in this case also $\langle S_{i+1}^j \rangle = \langle S_{i+1}^{j-1} \cup \{g_{i+1}\} \rangle < \langle S_i^j \rangle'$.

Case 4. $S_i^j = S_i^{j-1} \cup \{g_i\}$; $S_{i+1}^j = S_{i+1}^{j-1} \cup \{g_{i+1}\}$. From induction hypothesis, we have $\langle S_{i+1}^{j-1} \rangle < \langle S_i^{j-1} \rangle'$. It follows that $\langle S_{i+1}^{j-1} \rangle < \langle S_i^{j-1} \cup \{g_i\} \rangle' = \langle S_i^j \rangle'$. Now we are in the same situation as in Case 3, and using an argument identical to that of Case 3 we have $\langle S_{i+1}^j \rangle < \langle S_i^j \rangle'$.

Hence the claim. □

CLAIM 3.2.3. *In* CANONICALGENERATOR*, the input $\langle i, S_i^j, S_{i+1}^j, \ldots, S_{p(m)}^j \rangle$ to any stage j is such that for all $i < t \leq p(m)$, S_t^j is a canonical generator for the solvable group $\langle S_t^j \rangle$.*

Proof. We prove this by induction. For $j = 1$, it is easily verified that the claim is true. Let us assume that the claim is true for the jth stage. Let $\langle i, S_i^j, \ldots, S_{p(m)}^j \rangle$ be the input to the jth stage. Suppose the **while**-loop is exited through line 14 after l iterations with the value of g = YES. (If the loop is exited through line 11, then there are no more stages to be considered.) Then the value of $k = i + l$, and for $t > k$, S_t^j is not updated inside the loop, and hence by induction hypothesis it remains a canonical generator set for the solvable group $\langle S_t^j \rangle$. Since the value of $g = CheckCommutator(0^m, S_k^j, S_{k+1}^j)$ is YES we have that $\langle S_k^j \rangle' < \langle S_{k+1}^j \rangle$. From Claim 3.2.2 we have $\langle S_{k+1}^j \rangle < \langle S_k^j \rangle'$. Hence $\langle S_{k+1}^j \rangle = \langle S_k^j \rangle'$. It follows that $\langle S_k^j \rangle$ is solvable and S_t^j for $k \leq t \leq p(m)$ are generator sets for the commutator series of $\langle S_k^j \rangle$. Hence at line 18, CANONIZE will output a canonical generator set for each of the elements in the commutator series of $\langle S_k^j \rangle$. At line 19, i is updated to $k-1$, and the input to the $j+1$th stage is $\langle k-1, S_{k-1}^{j+1}, S_k^{j+1}, \ldots, S_{p(m)}^{j+1} \rangle$, where S_t^{j+1} is a canonical generator set for the solvable group $\langle S_t^{j+1} \rangle$ for $k \leq t \leq p(m)$. Hence the claim.[2] □

CLAIM 3.2.4. *In the algorithm* CANONICALGENERATOR*, for any stage j, it holds that $\exists i$ such that $|\langle S_i^{j+p(m)} \rangle| \geq 2|\langle S_i^j \rangle|$ if stage $j + p(m)$ exists.*

[2]Technically, one should establish Claims 3.2.2 and 3.2.3 together. We separated them for clarity of presentation.

Proof. Notice that if the algorithm at stage j enters the **while**-loop, then $\exists i$ such that $S_i^{j+1} = S_i^j \cup g$ for a $g \notin \langle S_i^j \rangle$, and the claim follows from Lagrange's theorem. So, it is enough to show that the **while**-loop is entered at least once after every $p(m)$ stages, if such a stage exists. Suppose the stage j is entered with the value of $i = i'$. Note that in a stage where the algorithm does not enter the **while**-loop, the variable k remains at i. So if the algorithm never enters the **while**-loop in the next $p(m)$ stages, at stage $(j + p(m) + 1)$, the value of $k = i = i' - (p(m) + 1) < 0$ for all $i' \leq p(m)$. But when $k = 0$ (line 15) the algorithm stops. Therefore it is not possible that $k < 0$. Hence the claim. □

To complete the proof of the theorem, first we shall see that the algorithm CanonicalGenerator runs in time polynomial in the length of the input. Observe that it is enough to show that the number of stages executed by the algorithm is bounded by a polynomial, since the number of iterations of the **while**-loop in lines 7–14 is bounded by $p(m)$. Now, the claim is that the number of stages executed by the algorithm is bounded by $2p^3(m)$. First, notice that for any $H < B_m$, $|H| \leq 2^{p(m)}$. Hence for any j, $\prod_{i=1}^{p(m)} |\langle S_i^j \rangle| \leq 2^{p^2(m)}$. Suppose the claim is false. Now from Claim 3.2.4 it follows that $\prod_{i=1}^{p(m)} |\langle S_i^{j+p(m)} \rangle| \geq 2 \prod_{i=1}^{p(m)} |\langle S_i^j \rangle|$. Hence $\prod_{i=1}^{p(m)} |\langle S_i^{2p^3(m)} \rangle| > 2^{p^2(m)}$, a contradiction.

Now we shall see that CanonicalGenerator makes only 1-guarded queries to L_{ca}, where $L_{ca} = L' \oplus L$. Let us first see that the queries to L through CheckCommutator are 1-guarded. It is enough to show that whenever CanonicalGenerator calls CheckCommutator with argument $(0^m, S_k^j, S_{k+1}^j)$ in *stage* j, S_{k+1}^j is a canonical generator set. But from Claim 3.2.3, the input $\langle i, S_i^j, S_{i+1}^j, \ldots, S_{p(m)}^j \rangle$ to any stage j is such that for all $i < t \leq p(m)$, S_t^j is a canonical generator for the solvable group $\langle S_t^j \rangle$. In this stage CanonicalGenerator calls CheckCommutator only with argument $(0^m, S_k^j, S_{k+1}^j)$ for $k \geq i$. Since S_{k+1}^j for $k \geq i$ is a canonical generator set, the queries made to L will be 1-guarded.

To see that the queries to L' through Canonize are 1-guarded, notice that calls to Canonize are made outside the **while**-loop. This means that CheckCommutator with input $(0^m, S_k^j, S_{k+1}^j)$ returns YES. That is, $\langle S_k^j \rangle' < \langle S_{k+1}^j \rangle$. Hence $\langle S_k^j \rangle' = \langle S_{k+1}^j \rangle$ from Claim 3.2.2. So it follows that calls to Canonize with argument $(S_i^j, S_{i+1}^j, \ldots, S_{p(m)}^j)$ will be such that S_l^j for $i \leq l \leq p(m)$ will generate the commutator series of S_i^j for all i. It follows from Theorem 2.11 that queries to L' will be 1-guarded.

Finally, we show that the above algorithm on input $(0^m, S)$ outputs a canonical generator set for the group $G = \langle S \rangle$ if G is solvable and outputs NOT SOLVABLE otherwise. Now, observe that if $H_1 < H_2$ are two finite groups, $H_1' < H_2'$. Hence it follows from Claim 3.2.2 that $\langle S_i^j \rangle < G_i$ for any i at any stage j, where G_i is the ith element in the commutator series of G. We know that after the execution of $2p^3(m)$ stages, the algorithm outputs either a set $X \subseteq B_m$ or NOT SOLVABLE. Suppose it outputs NOT SOLVABLE in stage j. This happens after the value of the variable inside the **while**-loop is assigned $p(m)$. From line 8 of the algorithm it follows that $S_{p(m)}^j \neq \{e\}$. But if G where solvable, then we know that $G_{p(m)} = \{e\}$, and since $\langle S_{p(m)}^j \rangle < G_{p(m)}$ from Claim 3.2.2, we have a contradiction.

Suppose the algorithm outputs a set $X \subseteq B_m$ at line 16 in stage j. Thus the value of the variable k is 0. Notice that inside the **while**-loop, the value of k is only incremented. This implies that at stage j the **while**-loop is not entered (the value of i could not have become 0 at a previous stage). So input to stage j is

$\langle 0, S_0^j, \ldots, S_{p(m)}^j \rangle$. From Claim 3.2.3, it follows that for all $1 \leq t \leq p(m)$, $\langle S_t^j \rangle$ is solvable and S_t^j is a canonical generator set for the group $\langle S_t^j \rangle$. From the value of g = YES and Claim 3.2.2, it follows that $\langle S_1^j \rangle' = \langle S_2^j \rangle$. Also, since $S_0^j = S$ for any stage j, it follows that S_i^j generates the ith element in the commutator series of $\langle S \rangle = G$. Hence, from Theorem 2.11, it follows that $[Canonize(S_1^j, \ldots, S_{p(m)}^j)]_l$ is a canonical generator set for G_l. Hence the theorem. □

4. Applications to computational problems. In this section we will use the algorithm CanonicalGenerator to prove upper bounds on the complexity of computational problems over solvable black-box groups that we defined in section 2.

Theorem 4.1. *Over any group family,* solvability testing *is in* SPP *and hence low for all gap-definable counting classes.*

Proof. We can easily modify CanonicalGenerator to get an oracle machine which makes only 1-guarded queries to an NP oracle to test whether the input group is solvable or not. Now applying Corollary 2.4 we get that solvability testing is in SPP. □

Remark. In [5], a co-RP algorithm is given for solvability testing. While this upper bound gives lowness for PP [17], it does not show lowness for other gap-definable counting classes.

In view of the above theorem and the fact that $\mathrm{P}^{\mathrm{SPP}} = \mathrm{SPP}$, for all the problems that we consider here, we assume without loss of generality that the groups encoded in the problem instances are solvable.

Theorem 4.2. *Over any group family,* membership testing *for the subclass of solvable groups is in* SPP *and hence low for all gap-definable counting classes.*

Proof. A direct application of Theorem 3.1, Proposition 2.8, and Corollary 2.4 shows that membership testing over solvable groups is in SPP. □

Theorem 4.3. *Over any group family,* subgroup testing *for the subclass of solvable groups is in* SPP *and hence low for all gap-definable counting classes.*

Proof. Let S_1 and S_2 be generator sets for groups G_1 and G_2. Then G_1 is a subgroup of G_2 if and only if for all $g \in S_1$, $g \in \langle S_2 \rangle$. This testing can be done in $\mathrm{P}^{\mathrm{SPP}}$ since membership testing is in SPP. Since $\mathrm{P}^{\mathrm{SPP}} = \mathrm{SPP}$, it follows that subgroup testing is in SPP. □

Theorem 4.4. *Over any group family,* normality testing *for the subclass of solvable groups is in* SPP *and hence low for all gap-definable counting classes.*

Proof. Let S_1 and S_2 be generator sets for groups G_1 and G_2. In order to check whether G_2 is normal in G_1 it is enough to check for pairs $g_1 \in S_1$ and $g_2 \in S_2$ whether $g_1 g_2 g_1^{-1} \in G_2$. This can be done in $\mathrm{P}^{\mathrm{SPP}}$ since membership testing is in SPP. Since $\mathrm{P}^{\mathrm{SPP}} = \mathrm{SPP}$, it follows that normality testing is in SPP. □

We have seen that checking whether a group is solvable or not is in SPP. Using this result we can design an SPP algorithm for testing whether a group is nilpotent or not. We need the following generalization of Theorem 3.2. We omit the proof of this group-theoretic result.

Theorem 4.5. *Let G be a group and let H_1 and H_2 be normal subgroups of G generated by S_1 and S_2. Then the mutual commutator subgroup $[H_1, H_2]$ is the normal closure of the set $\{g_1 g_2 g_1^{-1} g_2^{-1} \mid g_1 \in S_1, g_2 \in S_2\}$.*

Using this result we can design an oracle algorithm for computing a generator set for the mutual commutator subgroup for a given pair of groups. The following is the subroutine for computing a generator set for the mutual commutator subgroup. We will be interested in the case when the groups involved are solvable. This subroutine

will use membership testing, normality testing, and solvability testing as oracles.

MutualCommutator$(0^m, S, S_1, S_2)$
1 **if** $\langle S\rangle$ or $\langle S_1\rangle$ or $\langle S_2\rangle$ not Solvable
2 **then Output** NA
3 **if** $\langle S_1\rangle$ is not Normal in $\langle S\rangle$ or $\langle S_2\rangle$ is not Normal in $\langle S\rangle$
4 **then Output** NA
5 $X \leftarrow \{ghg^{-1}h^{-1} \mid g \in S_1, h \in S_2\}$
6 **while** $\exists g \in S$; $x \in X$ such that $(0^m, X, gxg^{-1}) \notin$ membership testing
7 **do** $X \leftarrow X \cup gxg^{-1}$
8 **end-while**
9 **Output** X

Theorem 4.6. *Over any group family,* nilpotence testing *is in* SPP *and hence low for all gap-definable counting classes.*

Proof. We can use the subroutine MutualCommutator for designing the following $\mathrm{P}^{\mathrm{SPP}}$ algorithm for nilpotence testing.

NilpotenceTester$(0^m, S)$
1 **if** $\langle S\rangle$ is not Solvable
2 **then Output** NOT NILPOTENT
3 $L^0 \leftarrow S$
4 **for** $i = 1$ to $p(m)$
5 **do** $L^i \leftarrow MutualCommutator(0^m, S, S, L^{i-1})$
6 **if** $\langle L^i\rangle = \langle L^{i-1}\rangle \neq \{e\}$
7 **then Output** NOT NILPOTENT
8 **end-for**
9 **Output** NILPOTENT

The above algorithm first tests whether the group is solvable. Since any nilpotent groups is also solvable, in the remaining computation we need only worry about groups which are solvable. Hence we can use any of the problems that we have already shown to be in SPP as oracles. The algorithm essentially computes the lower central series of the group. If the lower central series is not terminating, then for some i between 1 and $p(m)$ (where p is the polynomial bounding the size of B_m in the definition of the group family) the mutual commutator subgroups L^i and L^{i-1} must be equal and nontrivial for some i. This is tested in line 6. Note that testing whether two solvable groups are equal can be done with two queries to the subgroup testing oracle which we have shown to be in SPP. Hence the overall algorithm is a $\mathrm{P}^{\mathrm{SPP}}$ algorithm which can be converted into an SPP algorithm. □

Theorem 4.7. *Over any group family,* order verification *for the subclass of solvable groups is in* SPP *and hence low for all gap-definable counting classes.*

Proof. Let S be the generator for a solvable group G. Let the sequence $G = G_0 > G_1 > \cdots > G_k = \{e\}$ be the commutator sequence of G. Then we have $|G| = \prod_{i=1}^{k} |\frac{G_{i-1}}{G_i}|$. Consider an oracle machine M_o which uses oracle L_o as follows. M_o first simulated CanonicalGenerator to construct a canonical generator set for G. Then it simulates the oracle algorithm M_f provided in Lemma 2.10 to compute $|\frac{G_{i-1}}{G_i}|$ for each i using the canonical generator for G_{i-1} and G_i. Finally, it computes $\prod_{i=1}^{k} |\frac{G_{i-1}}{G_i}|$. The oracle L_o is the disjoint union of the NP oracles used by CanonicalGenerator and M_f. CanonicalGenerator makes only 1-guarded queries. Also since inputs

to M_f are canonical generator sets, M_f makes only 1-guarded queries. Therefore M_o makes only 1-guarded queries to L_o. Hence by Corollary 2.4, order verification for the subclass of solvable groups is in SPP. □

THEOREM 4.8. *Over any group family,* group isomorphism *for the subclass of solvable groups is in* SPP *and hence low for all gap-definable counting classes.*

Proof. We know that two solvable groups G and H are isomorphic if and only if the factor group H_{i-1}/H_i is isomorphic to G_{i-1}/G_i for all i where H_i (G_i) is the ith element in the commutator series of H (respectively, G). Isomorphism among abelian groups can be tested by computing the orders of the elements in the independent generators. Two abelian groups are isomorphic if the orders of the elements in their independent generators (arranged in some fixed order, say ascending) are the same. This suggests the following oracle algorithm M_{is} for testing isomorphism. M_{is} uses a language $L_{is} \in$ NP as oracle.

Let S_1 and S_2 be the generator sets for solvable groups G and H. M_{is} first computes canonical generator sets for both G and H using CANONICALGENERATOR. Then it simulates the oracle algorithm M_f provided in Lemma 2.10 to compute independent generators sets for the abelian factor groups H_{i-1}/H_i and G_{i-1}/G_i for all i. For each of these groups it computes the orders of all the elements in the independent generator set, say by using M_f. Then it compares whether the orders of the elements of the independent generator of H_{i-1}/H_i is the same as orders of the elements of the independent generator of G_{i-1}/G_i for all i to decide whether G and H are isomorphic. The oracle L_{is} is the disjoint union of the NP oracles used by CANONICALGENERATOR and M_f. CANONICALGENERATOR makes only 1-guarded queries. Also since inputs to M_f are canonical generator sets, M_f makes only 1-guarded queries. Therefore M_{is} makes only 1-guarded queries to L_{is}. Hence by Corollary 2.4, group isomorphism for the subclass of solvable groups are in SPP. □

For proving upper bound for group intersection, we need a result from [3], which we state next as a lemma.

LEMMA 4.9 (see [3, Lemma 5.7]). *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. There is a deterministic polynomial time oracle machine M that takes $(0^m, X, Y, Z, K)$ as input. Let $G = \langle X \rangle$ and $H = \langle Y \rangle$. Suppose the input has the following properties:*

1. *G and H are solvable groups.*
2. *X and Y are canonical generator sets for G and H, respectively.*
3. *Z is a canonical generator set for G' the commutator subgroup for G.*
4. *The number K is $|G' \cap H|$.*

Then the machine M outputs $|G \cap H|$. Furthermore, M makes $|B_m|$-guarded queries to an NP *language A and* 1*-guarded queries to another* NP *language B. The behavior of machine M is not specified for inputs that does not satisfy the above properties.*

THEOREM 4.10. *Over any group family,* group intersection *for the subclass of solvable groups is in* LWPP *and hence low for the classes* PP *and* $C_=P$.

Proof. Let S_1 and S_2 be the generator sets for solvable groups G and H. We will design an oracle algorithm M_{int} which first computes canonical generator sets for all the groups in the commutator series of both G and H, and then uses the machine M from the above result inductively to compute $G \cap H$ as follows. Let $G = G_0 > G_1 > \cdots > G_{k-1} > G_k = \langle e \rangle$ be the commutator series for G. To begin with, G_{k-1} is abelian, and we have an independent generator set for it. Applying the algorithm M of Lemma 4.9 to the groups G_{k-1} and H, with $Z = \{e\}$ and $K = 1$, M_{int} can compute $|G_{k-1} \cap H|$. Now, M can be used inductively to compute $|G_{i-1} \cap H|$

given canonical generator sets for G_i and H as well as the number $|G_i \cap H|$ for each i. Since all the generator sets that input to M are canonical generator sets, it is clear that M_{int} makes only $|B_m|$-guarded and 1-guarded queries. Also by standard techniques, 1-guarded queries can be made $|B_m|$-guarded. Now applying Corollary 2.4 we get that group intersection is in LWPP. □

Finally we show that group factorization, coset intersection, and double coset memb for solvable groups are also in LWPP.

THEOREM 4.11. *Over any group family, the problems* group factorization, coset intersection, *and* double coset memb *for the subclass of solvable groups are in* LWPP *and hence low for the classes* PP *and* $C_=P$.

Proof. We give proof for group factorization. Proofs for the other two problems are very similar and are omitted. Let $\mathcal{B} = \{B_m\}_{m \geq 1}$ be a group family with elements of B_m uniquely encoded as strings in $\Sigma^{p(m)}$ for a polynomial p. Consider the following NP machine R, which takes $\langle 0^m, S_1, S_2, g, K \rangle$ as input, where S_1 and S_2 are the generator sets for two groups, g is an element of B_m, and K divides $|B_m|$.

R$(0^m, S_1, S_2, g, K)$
1 Verify that K divides $|B_m|$;
2 Guess $w \in \Sigma^{p(m)}$;
3 Verify that $w \in \langle S_1 \rangle$ and $w^{-1}g \in \langle S_2 \rangle$;
4 Branch into $|B_m|/K$ paths and **accept** on each of them.

We are interested in the behavior of this machine only when the following conditions are satisfied: S_1 and S_2 are canonical generator sets for two solvable groups G and H, and $K = |G \cap H|$. Line 3 of this algorithm can be implemented by simulating M_u in Proposition 2.8.

Machine R accepts the input if and only if $(0^m, S_1, S_2, g)$ is a positive instance of group factorization. This is because $g \in GH$ if and only if there exists a w such that $w \in G$ and $w^{-1}g \in H$. Moreover, in this case R will have $\frac{|B_m|}{K} * |G \cap H|$ accepting paths since there will be exactly $|G \cap H|$ elements w satisfying $w \in G$ and $w^{-1}g \in H$. Now consider an oracle machine M_{gf} which on input $(0^m, S_1, S_2, g)$ as input, first computes canonical generator sets for G and H using CANONICALGENERATOR, then computes $|G \cap H|$ using the machine M_{int}, and then queries the language accepted by R the string $(0^m, S_1, S_2, g, |G \cap H|)$. It is clear that M_{gf} only makes $|B_m|$-guarded and 1-guarded queries. 1-guarded queries can be made $|B_m|$-guarded using standard techniques. Therefore, applying Corollary 2.4, we have that group factorization is in LWPP. □

5. Conclusion and open problems. In this paper we investigated the complexity of many computational group-theoretic problems over solvable black-box groups. All the problems considered are computationally difficult: there is no known polynomial time algorithm for any of them. But it is also not known whether they are hard for NP. We showed that these problems over solvable black-box groups are in the counting class SPP or LWPP. The SPP upper bound implies that they will not provide additional power as oracles to counting classes such as PP, $C_=P$, and Mod_kP. The upper bound of LWPP implies that problems will not provide additional power as oracles to counting classes such as PP and $C_=P$. These upper bounds support the belief that the problems considered are unlikely to be hard for NP. Another important aspect of the results is that they provide more examples of natural problems in SPP and LWPP which are not known to be in P.

There is one major open question that arises from this investigation: extend the above upper bound to problems defined over a general black-box group. We believe that the problems we considered here are in SPP even over general black-box groups. The methods we used for solvable groups will not extend for the general case. We feel that new methods based on classification theorems about finite simple groups may be required for the proof of this. A more general question is to show membership of other natural problems in the counting complexity classes like SPP or LWPP.

Acknowledgments. I thank V. Arvind for discussions we had about the results in this paper and his comments. I would like to thank the referees for carefully reading the first version of this paper and giving very detailed comments and pointing out crucial omissions. Their input has helped me to greatly improve the presentation of the paper.

REFERENCES

[1] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, manuscript; available online from http://citeseer.nj.nec.com/agrawal02primes.html.

[2] V. Arvind and P. Kurur, *Graph isomorphism is in SPP*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2002, pp. 743–750.

[3] V. Arvind and N. V. Vinodchandran, *Solvable black-box group problems are low for PP*, Theoret. Comput. Sci., 180 (1997), pp. 17–45.

[4] L. Babai, *Bounded round interactive proofs in finite groups*, SIAM J. Discrete Math., 5 (1992), pp. 88–111.

[5] L. Babai, G. Cooperman, L. Finkelstein, E. Luks, and Á. Seress, *Fast Monte Carlo algorithms for permutation groups*, J. Comput. System Sci., 50 (1995), pp. 296–308.

[6] L. Babai and M. Szemerédi, *On the complexity of matrix group problems* I, in Proceedings of the 25th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1984, pp. 229–240.

[7] J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity.* I, Springer-Verlag, Berlin, Hiedelberg, 1988.

[8] J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity.* II, Springer-Verlag, Berlin, Hiedelberg, 1990.

[9] R. Boppana, J. Hastad, and S. Zachos, *Does co-NP have short interactive proofs?*, Inform. Process. Lett., 25 (1987), pp. 127–132.

[10] W. Burnside, *Theory of Groups of Finite Order*, Dover, New York, 1955.

[11] G. Cooperman and L. Finkelstein, *Random algorithms for permutation groups*, CWI Quarterly, 5 (2) (1992), pp. 93–105.

[12] S. Fenner, L. Fortnow, and S. Kurtz, *Gap-definable counting classes*, J. Comput. System Sci., 48 (1994), pp. 116–148.

[13] M. Hall, *The Theory of Groups*, Macmillan, New York, 1959.

[14] C. Hoffmann, *Group-Theoretic Algorithms and Graph Isomorphism*, Lecture Notes in Comput. Sci. 136, Springer-Verlag, Berlin, 1982.

[15] C. Hoffmann, *Subcomplete generalizations of graph isomorphism*, J. Comput. System Sci., 25 (1982), pp. 332–359.

[16] J. Köbler, U. Schöning, and J. Torán, *Graph isomorphism is low for PP*, J. Comput. Complexity, 2 (1992), pp. 301–310.

[17] J. Köbler, U. Schöning, S. Toda, and J. Toran, *Turing machines with few accepting computations and low sets for PP*, J. Comput. System Sci., 44 (1992), pp. 272–286.

[18] E. Luks, *Isomorphism of graphs of bounded valence can be tested in polynomial time*, J. Comput. System Sci., 25 (1982), pp. 42–65.

[19] U. Schöning, *Graph isomorphism is in the low hierarchy*, J. Comput. System Sci., 37 (1988), pp. 312–323.

[20] N. V. Vinodchandran, *Improved lowness results for solvable black-box group problems*, in Proceedings of the 17th International Conference on the Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 1346, Springer-Verlag, Berlin, 1997, pp. 220–234.

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 870–891

TIME OF DETERMINISTIC BROADCASTING IN RADIO NETWORKS WITH LOCAL KNOWLEDGE*

DARIUSZ R. KOWALSKI† AND ANDRZEJ PELC‡

Abstract. We consider broadcasting in radio networks, modeled as undirected graphs, whose nodes know only their own label and labels of their neighbors. In every step every node acts either as a *transmitter* or as a *receiver*. A node acting as a transmitter sends a message which can potentially reach all of its neighbors. A node acting as a receiver in a given step gets a message if and only if exactly one of its neighbors transmits in this step.

Bar-Yehuda, Goldreich, and Itai [*J. Comput. System Sci.*, 45 (1992), pp. 104–126] considered broadcasting in this model. They claimed a linear lower bound on the time of deterministic broadcasting in such radio networks of diameter 3. This claim turns out to be incorrect in this model (although it is valid in a more pessimistic model [R. Bar-Yehuda, O. Goldreich, and A. Itai, *Errata Regarding "On the time complexity of broadcast in radio networks: An exponential gap between determinism and randomization,"* http://www.wisdom.weizmann.ac.il/mathusers/oded/p_bgi.html, 2002]). We construct an algorithm that broadcasts in logarithmic time on all graphs from the Bar-Yehuda, Goldreich, and Itai paper (BGI). Moreover, we show how to broadcast in sublinear time on all n-node graphs of diameter $o(\log\log n)$. On the other hand, we construct a class of graphs of diameter 4, such that every broadcasting algorithm requires time $\Omega(\sqrt[4]{n})$ on these graphs. In view of the randomized algorithm from BGI, running in expected time $\mathcal{O}(D\log n+\log^2 n)$ on all n-node graphs of diameter D (cf. also a recent $\mathcal{O}(D\log(n/D)+\log^2 n)$-time algorithm from [D. Kowalski and A. Pelc, *Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing*, Boston, 2003, pp. 73–82; A. Czumaj and W. Rytter, *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, Cambridge, MA, 2003, pp. 492–501]), our lower bound gives the first correct proof of an exponential gap between determinism and randomization in the time of radio broadcasting, under the considered model of radio communication.

Key words. broadcasting, distributed, deterministic, radio network

AMS subject classifications. 68W15, 68Q25, 68Q17

DOI. 10.1137/S0097539702419339

1. Introduction.

A radio network is modeled as an undirected connected graph whose nodes are transmitter-receiver devices. An edge e between two nodes means that the transmitter of one end of e can reach the other end. Nodes send messages in synchronous *steps* (time slots), measured by a global clock which indicates the current step number. In every step every node acts either as a *transmitter* or as a *receiver*. A node acting as a transmitter sends a message which can potentially reach all of its neighbors. A node acting as a receiver in a given step gets a message if and only if exactly one of its neighbors transmits in this step. The message received in this case is the one that was transmitted. If at least two neighbors of u transmit simultaneously

*Received by the editors December 7, 2002; accepted for publication (in revised form) November 12, 2003; published electronically May 5, 2004. A preliminary version of this paper appeared in the Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002), November 2002, Vancouver, Canada, under the title "Deterministic Broadcasting Time in Radio Networks of Unknown Topology."

http://www.siam.org/journals/sicomp/33-4/41933.html

†Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02-097 Warszawa, Poland, and Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, Saarbrücken, 66123 Germany (darek@mimuw.edu.pl). This work was done in part during this author's stay at the Research Chair in Distributed Computing of the Université du Québec en Outaouais, as a postdoctoral fellow. This author's research was supported in part by KBN grant 4T11C04425.

‡Département d'informatique, Université du Québec en Outaouais, Hull, QC J8X 3X7, Canada (pelc@uqo.ca). This author's research was supported in part by NSERC grant OGP 0008136 and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

in a given step, none of the messages is received by u in this step. In this case we say that a *collision* occurred at u. It is assumed that the effect at node u of more than one of its neighbors transmitting is the same as that of no neighbor transmitting (i.e., a node cannot distinguish a collision from silence). We assume that nodes know only their own label and the labels of their neighbors. Apart from that, the only a priori information on the network available to nodes is a polynomial upper bound on the number of nodes.

One of the fundamental tasks in network communication is *broadcasting*. Its goal is to transmit a message from one node of the network, called the *source*, to all other nodes. Remote nodes get the source message via intermediate nodes, along paths in the network. In this paper we concentrate on one of the most important and widely studied performance parameters of a broadcasting scheme, which is the total time—that is, the number of steps it uses to inform all the nodes of the network. We measure complexity in terms of the number of nodes in the network.

1.1. Our results. In a seminal paper [3], Bar-Yehuda, Goldreich, and Itai considered broadcasting in radio networks in the model described above. They claimed a linear lower bound on the time of deterministic broadcasting in such radio networks of diameter 3. This claim turns out to be incorrect, although it is valid in a more pessimistic model [4]. In fact, as pointed out in [4], the error is due to a gap between two models handling collisions in radio broadcasting: that from [3] and a more pessimistic but equally reasonable one. (See more comments on this point in subsection 1.2.) As discussed below, a lot of work on radio broadcasting has been done following [3], most of it modeling collisions as in [3].

Using the same model as in [3], we construct an algorithm that broadcasts in logarithmic time on all graphs from [3]. Moreover, we show how to broadcast in sublinear time on all n-node graphs of diameter $o(\log \log n)$. On the other hand, we construct a class of graphs of diameter 4, such that every broadcasting algorithm requires time $\Omega(\sqrt[4]{n})$ on one of these graphs. In view of the randomized algorithm from [3] running in expected time $\mathcal{O}(D \log n + \log^2 n)$ on all n-node graphs of diameter D (cf. also a recent $\mathcal{O}(D \log(n/D) + \log^2 n)$-time algorithm from [23, 15]), our lower bound gives the first correct proof of an exponential gap between determinism and randomization in the time of radio broadcasting, in the model from [3].

1.2. Related work. Most of the results concerning broadcasting in radio networks can be divided into two parts: those which assume complete knowledge of the topology of the network at all nodes, or equivalently, dealing with centralized broadcasting for a given network, and those assuming only limited knowledge of the network at all nodes and dealing with distributed broadcasting in arbitrary networks.

Deterministic centralized broadcasting assuming complete knowledge of the network was first considered in [10], where a $\mathcal{O}(D \log^2 n)$-time broadcasting algorithm was given for all n-node networks of diameter D. In [18], $\mathcal{O}(D + \log^5 n)$-time broadcasting was proposed. On the other hand, in [1] the authors proved the existence of a family of n-node networks of radius 2, for which any broadcast requires time $\Omega(\log^2 n)$.

The study of deterministic distributed broadcasting in radio networks whose nodes have only limited knowledge of the topology was initiated in [3]. The authors assumed that nodes know only their own label and the labels of their neighbors (and that collision at a node has the same effect as silence). Under this scenario, a simple $\mathcal{O}(n)$-time broadcasting algorithm based on depth-first search (DFS) follows from [2]. In [3] the authors constructed a class of n-node graphs of diameter 3, and claimed

that every broadcasting algorithm requires time $\Omega(n)$ on one of these graphs. In [19] this claim was further strengthened to a lower bound of $n-1$ on broadcasting time required on one of these graphs. It follows from the present paper that both these claims (concerning lower bounds) are incorrect.

As pointed out in [4], the linear lower bound from [3] is valid in a more pessimistic model than that of [3]. Namely, one could assume that in the case of a collision at u the effect can be either the same as if no neighbor of u transmitted or the same as if *any single* neighbor of u transmitted, the choice of the effect being left to the adversary. That is, either noise caused by many transmitting neighbors may be undistinguishable from background noise or else one of the competing neighbors may prevail, and it is impossible to predict which situation occurs for a given collision. For this model, which seems equally reasonable to that from [3], the argument and the linear lower bound of [3] are valid (cf. [4]). In fact, as explained in [4], the error in [3] is due to the gap between these two models.

Many authors [5, 7, 8, 9, 11, 12, 13, 14, 16, 26] studied deterministic distributed broadcasting in radio networks under the even weaker assumption that nodes know only their own label (but do not know the labels of their neighbors). In all these papers the collision issue was modeled as in [3]. In [11] the authors gave a broadcasting algorithm working in time $\mathcal{O}(n)$ for arbitrary n-node networks, assuming that nodes can transmit spontaneously before getting the source message. It was shown in [24] that if nodes know only their own label, the argument from [3] can be modified to prove a lower bound $\Omega(n)$ on broadcasting time for networks of radius 2. Thus the algorithm from [11] is optimal.

In [11, 12, 13, 14, 16, 26] the model of directed graphs was used. Increasingly faster broadcasting algorithms working on arbitrary n-node (directed) radio networks were constructed, culminating with the $\mathcal{O}(n\log^2 n)$-time algorithm from [13]. Recently, a $\mathcal{O}(n\log n\log D)$-time broadcasting algorithm was shown in [22] for n-node networks of radius D. This was further improved to $\mathcal{O}(n\log^2 D)$ in [15]. In [14] the authors showed a lower bound $\Omega(n\log D)$ on broadcasting time for n-node networks of radius D. On the other hand, in [5, 7, 8, 9, 12, 14] the problem was to find efficient broadcasting algorithms on radio networks of maximum in-degree Δ.

Finally, randomized broadcasting algorithms in radio networks were studied, e.g., in [3, 15, 25, 23]. For these algorithms no topological knowledge of the network was assumed. In [3] the authors showed a randomized broadcasting algorithm running in expected time $\mathcal{O}(D\log n+\log^2 n)$. A faster algorithm, running in expected time $\mathcal{O}(D\log(n/D)+\log^2 n)$ was presented in [23] (see also [15]). In [25] it was shown that for any randomized broadcasting algorithm (and parameters $D\le n$), there exists an n-node network of diameter D requiring expected time $\Omega(D\log(n/D))$. It should be noted that the lower bound $\Omega(\log^2 n)$ from [1], for some networks of radius 2, holds for randomized algorithms as well. This shows that the algorithm from [23] is optimal.

1.3. Organization of the paper. In section 2 we summarize the communication model (taken from [3]) and the terminology used in this paper. Section 3 is devoted to showing a logarithmic broadcasting algorithm for the class of networks for which a linear lower bound was claimed in [3]. In this section we first describe the novel Procedure Echo, which is later used for more complicated algorithms. In section 4 we describe and analyze a broadcasting algorithm working in sublinear time on all shallow networks. This indicates that if a linear lower bound on broadcasting time can at all be proved (for networks of sublinear diameter), then it requires the construction of quite complicated networks. Section 5 is devoted to the proof of the

lower bound $\Omega(\sqrt[4]{n})$ on broadcasting time in networks of bounded diameter. Finally, section 6 contains concluding remarks and open problems.

2. Model and terminology. We consider undirected graphs whose nodes have distinct labels belonging to the set $\{0, 1, \ldots, r\}$, where r is polynomial in the number n of nodes. The parameter r is known to all nodes. In the lower bounds we assume that n itself is known to all nodes. A distinguished node with label 0 is called the *source*. We denote by D the *radius* of the graph, i.e., the distance from the source to the farthest node. (For undirected graphs, the diameter is of the order of the radius.) The jth *layer* of a graph is the set of nodes at distance j from the source. We adopt the communication model used by Bar-Yehuda, Goldreich, and Itai [3]. It is summarized in the following definition.

DEFINITION 2.1 (see [3]). *A* broadcast protocol *for radio networks is a multiprocessor (multinode) protocol, the execution of which proceeds in steps (time-slots) (numbered* $0, 1, \ldots$*) as follows:*

1. *In the initial step* 0*, only the source transmits a message called the* source message*.*
2. *In each step each node acts either as a transmitter or as a receiver (or is inactive).*
3. *A node receives a message in a specific step if and only if it acts as a receiver in this step and exactly one of its neighbors acts as a transmitter in that step. The message received in this case is the message transmitted by that neighbor.*
4. *The action of a node in a specific step is determined as a function of its initial input (which consists of its own label and the labels of its neighbors) and the (sequence of) messages that it has received in previous steps. All nodes have identical copies of the same program.*
5. *A node may act as a transmitter in a step* > 0 *only if it has received a message in a previous time-slot (there are no "spontaneous" transmissions of nodes other than the source in step* 0*).*
6. *The broadcast is completed at step* t *if all nodes have received the source message at one of the steps* $0, 1, \ldots, t$*.*

As in [3], we assume that a node cannot distinguish whether more than one neighbor or no neighbor transmitted in a given step; i.e., we work in the model without collision detection.

3. Logarithmic broadcasting in *BGI* networks. In [3] the following class of $(n+2)$-node networks was defined. Let S be a nonempty subset of $\{1, \ldots, n\}$. The network G_S is a graph (of radius 2) whose nodes are labeled $0, 1, \ldots, n+1$. The set of edges of G_S is $E = \{(0, i) : 1 \leq i \leq n\} \cup \{(i, n+1) : i \in S\}$. Node 0 is the source, and node $n+1$ (the only node in layer 2) is called the *sink*. We refer to all networks G_S as *BGI-n* networks (see Figure 3.1).

It was claimed in [3] that for any broadcast protocol there is a *BGI-n* network on which this protocol works in time $\Omega(n)$. However, assuming the model from [3], which is the same as that from the present paper, the proof in [3] contains the following flaw. *Predetermined* sets of nodes transmitting in consecutive steps are fixed, and then a network is constructed in which some node is not informed during any of these steps. However, during the broadcasting process, the source may potentially acquire some information, which it may pass to other nodes, thus modifying the sets of nodes transmitting in subsequent steps. So, in fact, under the considered model, the proof from [3] works only for oblivious algorithms, in which sets of nodes transmitting in a given step must be fixed in advance.

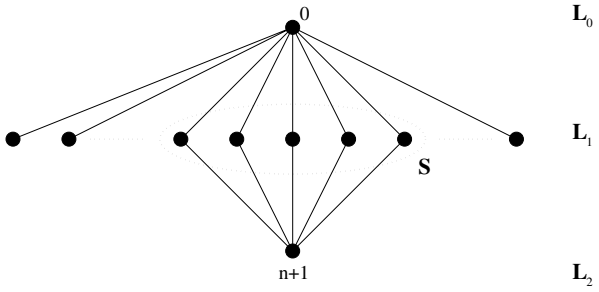


FIG. 3.1. *BGI-n network* G_S.

It turns out that not only is the argument from [3] erroneous under the considered model but, in fact, the above result itself is incorrect. (As mentioned in the introduction, the argument and the proof remain correct in a more pessimistic communication model.) Below we give an algorithm that broadcasts in all *BGI-n* networks in time $\mathcal{O}(\log n)$. The technique of selecting one out of many simultaneously transmitting neighbors, which is the main ingredient of this algorithm, will be further used in the construction of a much more involved algorithm which guarantees fast broadcasting in arbitrary networks of small radius.

The main idea of our algorithm is to simulate the collision detection capability in some nodes of the network. (Recall that collision detection is not available a priori in our model.) In order to get logarithmic broadcasting for *BGI-n* networks, it is enough to simulate collision detection at the source. To get sublinear broadcasting in all networks of radius $o(\log \log n)$, we will need to simulate this capability in many nodes from different levels.

Let A be a set of neighbors of the source (possibly unknown to it), and let $i \notin A$ be another neighbor of the source. Suppose that nodes in A want to transmit. Our goal is to let the source distinguish whether A has 0, 1, or more than 1 element. This can be done using the following 2-step procedure.

PROCEDURE ECHO (i, A).

Step 1. Every node in A transmits its label.

Step 2. Every node in $A \cup \{i\}$ transmits its label.

There are 3 possible effects of Procedure Echo (i, A) at the source.

Case 1. A message is received in Step 1 and no message in Step 2. In this case the source knows that A has 1 node and knows the label of this unique node.

Case 2. No message is received in Step 1 and a message (from i) is received in Step 2. In this case the source knows that A is empty.

Case 3. No message is received in either step. In this case the source knows that A has at least 2 nodes.

Procedure Echo is used to select one node in the set S of nodes connected to the sink, in a *BGI-n* network G_S. Once such a unique node is selected and transmits, the sink receives the source message, and broadcast is completed. This is done using the following algorithm. (In the original definition of *BGI-n* networks, nodes are labeled by consecutive numbers $0, 1, \ldots, n+1$, but we formulate our algorithm in the more general case, when labels are chosen arbitrarily from a set $\{0, 1, \ldots, r\}$, where r is polynomial in the number of nodes. Without loss of generality, assume that r is a power of 2.)

ALGORITHM BINARY-SELECTION-BROADCAST. In step 0, the source transmits the source message and the lowest label i of its neighbor (in the original definition of

BGI-n networks, $i = 1$). In step 1, node with label i transmits the source message and its degree. If this degree is 2 ($i \in S$), the sink receives the message and broadcast stops. Assume that the degree of i is 1.

All remaining steps $2, 3, \ldots$ are divided into segments of length 3. In the first step of each segment, the source transmits a range R of labels and orders the execution of Procedure Echo $(i, R \cap S)$ during the last two steps of the segment. (Notice that all nodes from layer 1 know if they are in $R \cap S$.) In the first segment, $R := \{1, \ldots, r/2\}$. If a range $R = \{x, \ldots, y\}$ is transmitted in a given segment, the range to transmit in the next segment is chosen according to the three possible effects of Procedure Echo $(i, R \cap S)$, described above. In Case 1, the sink is informed and broadcast stops. In Case 2, $R := \{y+1, \ldots, y+(y-x+1)/2\}$. In Case 3, $R := \{x, \ldots, (y+x-1)/2\}$. □

THEOREM 3.1. *Algorithm Binary-Selection-Broadcast completes broadcasting in any BGI-n network in time* $\mathcal{O}(\log n)$.

Proof. Since the size of R transmitted in the ith segment is $r/2^i$, after at most $\log r \in \mathcal{O}(\log n)$ steps, the set $R \cap S$ contains exactly one node, and hence broadcast is completed in time $\mathcal{O}(\log n)$. □

4. Sublinear time broadcasting in networks of radius $o(\log \log n)$. In this section we construct a broadcasting algorithm working in time $o(n)$ on all n-node networks of radius $o(\log \log n)$. We will use the following results from the literature. The following two theorems assume that nodes know parameters r and d but do not assume any knowledge of the network topology.

THEOREM 4.1 (see [14]). *Consider a radio network modeled by an arbitrary graph* (V, E), *where* V *is a subset of* $\{1, \ldots, r\}$. *Let* A *and* B *be a partition of* V *such that all nodes in* A *have the same message* m. *Then there exists a protocol working in time* $\mathcal{O}(\min(r, d \log(r/d)))$, *which makes message* m *known to all nodes* $v \in B$ *having at least one and at most* d *neighbors in* A.

THEOREM 4.2 (see [17, 14]). *Given a radio network modeled by an arbitrary graph* (V, E), *where* V *is a subset of* $\{1, \ldots, r\}$ *and in which every node has a (possibly different) message, there exists a protocol working in time* $\mathcal{O}(\min(r, d^2 \log r))$, *upon the completion of which, every node of degree at most* d *learns the messages of all its neighbors.*

It should be mentioned that, while the protocols in the above theorems were obtained in a nonconstructive way, constructive counterparts of both these results (involving polynomial time local computations) are known and yield only slightly slower protocols. A constructive counterpart of Theorem 4.1 yielding a $\mathcal{O}(\min(r, d \cdot \text{polylog}\, r))$-time protocol follows from [20, 27], and a constructive counterpart of Theorem 4.2 yielding a $\mathcal{O}(\min(r, d^2 \log^2 r))$-time protocol follows from [21]. In our algorithm we use protocols from Theorems 4.1 and 4.2, but these constructive counterparts could be used as well, and our resulting broadcasting algorithm would still work in sublinear time.

The next result refers to radio networks of known topology.

THEOREM 4.3 (see [10]). *Consider a radio network modeled by an arbitrary graph* (V, E), *where* V *is a subset of* $\{1, \ldots, r\}$, *and assume that all nodes know the topology of the graph. Let* A *and* B *be a partition of* V *such that all nodes in* A *have the same message* m. *Then there exists a protocol working in time* $\mathcal{O}(\log^2 r)$, *which makes message* m *known to all nodes* $v \in B$ *having a neighbor in* A.

4.1. Broadcasting in networks of radius 2. We first describe a sublinear time broadcasting algorithm working for all networks of radius 2. More generally, in every network G, this algorithm informs all nodes in levels 1 and 2 in sublinear time.

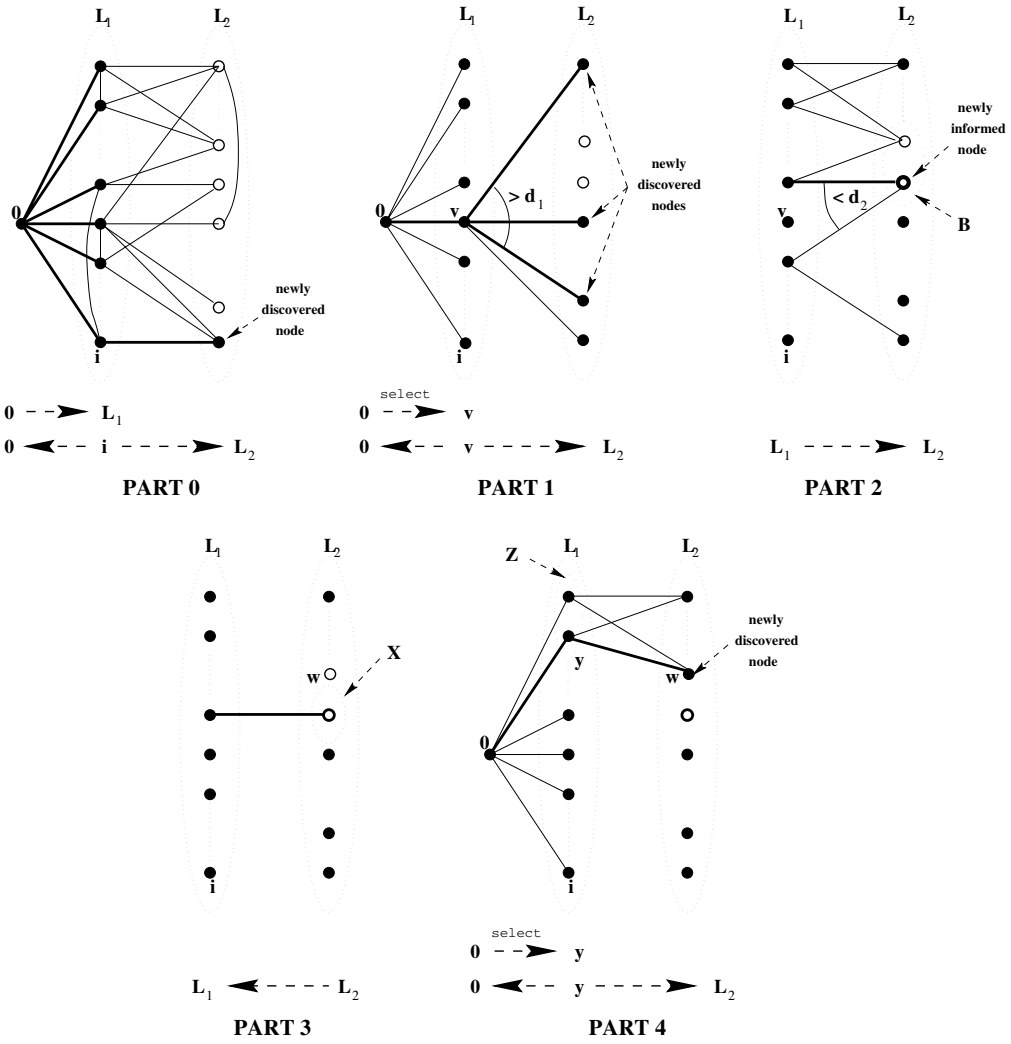


FIG. 4.1. *Algorithm* $\mathcal{A}_2$.

At each step of the execution, the source maintains a set DIS of *discovered* nodes: those about which it knows that they received the source message. The algorithm uses the polynomial upper bound r on the number of nodes, and two parameters d_1 and d_2, whose values will be specified later.

ALGORITHM $\mathcal{A}_2$ (see Figure 4.1).

Part 0. In step 0, the source transmits the source message, the set L_1, and the lowest label $i \in L_1$. In step 1, node with label i transmits the source message and the set C of its neighbors in L_2. The source sets the set DIS of discovered nodes to $\{0\} \cup L_1 \cup C$ and transmits it in step 2.

Part 1. Using a similar mechanism as in Algorithm Binary-Selection-Broadcast, the source selects a node in L_1 for which the number of undiscovered neighbors in L_2 is maximum. (Every node in L_1 can distinguish its neighbors in L_1 and in L_2.) In the next step the selected node transmits the source message and the set of its neighbors in L_2. The source adds them to discovered nodes and transmits the updated set DIS.

This process is repeated until there are no nodes in L_1 with more than d_1 undiscovered neighbors in L_2.

Part 2. Apply the protocol from Theorem 4.1 (with $d = d_2$) to the subgraph of G induced by $L_1 \cup B$ and to the partition (L_1, B), where B is the set of undiscovered nodes in L_2. This protocol makes the source message known to all undiscovered nodes from L_2 which have at most d_2 neighbors in L_1.

Part 3. Let X be the set of all nodes from L_2 which received the source message during Part 2 and have at most d_2 neighbors in L_1. Apply the protocol from Theorem 4.2 (for $d = d_1$) to the subgraph of G induced by $X \cup L_1$. The message transmitted by each node contains its label and the source message.

Part 4. All nodes from L_1 check if all their neighbors in L_2 got the source message. Nodes selected in Part 1 know that all their neighbors in L_2 were informed and discovered. Let Z be the set of nodes in L_1 which did not get a message in Part 3 from some of their undiscovered neighbors in L_2. Using a similar mechanism as in Algorithm Binary-Selection-Broadcast, the source selects one node in Z; then this node transmits (alone) the source message and the set of all its neighbors in L_2, and the source adds these neighbors to the set DIS of discovered nodes. After each selection, at least one currently undiscovered node gets the source message. This process continues until all nodes in L_1 know that all their neighbors in L_2 received the source message. □

THEOREM 4.4. *Algorithm $\mathcal{A}_2$ completes broadcasting in any n-node network of radius 2 in $\mathcal{O}(n^{2/3} \log n)$ time.*

Proof. Correctness is straightforward. The complexity of the algorithm is estimated as follows. Part 0 takes 3 steps. Each selection in Part 1 takes $\mathcal{O}(\log r)$ steps and there are at most n/d_1 selections; hence the entire Part 1 takes $\mathcal{O}((n/d_1) \cdot \log r)$ steps. In view of Theorem 4.1, Part 2 takes $\mathcal{O}(d_2 \cdot \log r)$ steps. In view of Theorem 4.2, Part 3 takes $\mathcal{O}(d_1^2 \log r)$ steps. Each selection in Part 4 takes $\mathcal{O}(\log r)$ steps and there are at most nd_1/d_2 selections; hence the entire Part 4 takes $\mathcal{O}((nd_1/d_2) \cdot \log r)$ steps. Taking $d_1 = \sqrt[3]{n}$ and $d_2 = \sqrt[3]{n^2}$, this adds up to $\mathcal{O}(n^{2/3} \log r) = \mathcal{O}(n^{2/3} \log n)$ steps. □

4.2. Extension to arbitrary networks of radius $o(\log \log n)$. We now describe an algorithm which broadcasts in sublinear time in arbitrary networks of radius $o(\log \log n)$. The algorithm uses the polynomial upper bound r on the number of nodes, and parameters d_j, d'_j, for $j = 2, 3, \ldots$, whose values will be specified later. The algorithm is constructed inductively. The construction is local, in the sense that every node constructs its part of the algorithm, using some coordination guaranteed by the properties and remarks listed below. After the kth phase of the algorithm, where k is an integer larger than 1, the following properties will be satisfied for some positive constant α.

Property 1. All nodes from layers L_j, $j = 1, \ldots, k$, know the source message and know to which layer they belong.

Property 2. For all $j = 1, \ldots, k$, a Procedure SEND(j), coordinated by the source, can be constructed, which has the following effect: if all nodes in L_{j-1} have the same message m and start at the same time, then all nodes in L_j learn message m. (SEND(1) consists of one step in which the source transmits.) Procedure SEND(j) lasts at most $\alpha \cdot (d_j + \log r) \log r$ time.

Remarks.

1. Let $j = 1, \ldots, k-1$, and $A \subseteq L_{j+1}$ be a set of nodes that want to transmit. It follows from Property 2 that a Procedure DETECT(j, A), coordinated by

the source, can be constructed, which enables every node in L_j to distinguish whether it has 0, 1, or more than 1 neighbor in A. Procedure DETECT(j, A) works as follows. Starting at a given time step t_0, all nodes in A repeat transmission in $1+\alpha\cdot(d_j+\log r)\log r$ consecutive time steps. Simultaneously, nodes in L_{j-1} perform Procedure SEND(j).
Each node $v \in L_j$ detects the number of its neighbors in A similarly as in Procedure Echo. By Property 2, v would get message m, during $\alpha\cdot(d_j+\log r)\log r$ steps after t_0, if nodes from A did not transmit. Hence v decides as follows. If it receives a message from a neighbor not in L_{j-1} during Procedure DETECT(j, A), it knows that it has exactly one neighbor in A and knows its label. Otherwise, two cases are possible. (1) If v receives only messages from L_{j-1} during Procedure DETECT(j, A), then it knows that none of its neighbors is in A. (2) If v receives no messages during Procedure DETECT(j, A), then it knows that at least two of its neighbors are in A.

2. Let $j = 0, \ldots, k-1$, $i \in L_j$, and let $A \subseteq L_{j+1}$ be a subset of neighbors of i that want to transmit. A Procedure SELECT(j, i, A), coordinated by node i, can be constructed, in which node i selects one element in A. This can be done in $\alpha \cdot \log r$ steps, similarly as in Algorithm Binary-Selection-Broadcast, where node i plays the role of the source.

ALGORITHM SUBLINEAR-BROADCAST. In phase 1 do nothing. In phase 2 perform algorithm $\mathcal{A}_2$. Let $k > 1$. Suppose that Properties 1 and 2 are satisfied after phase k. We describe phase $k+1$. The source maintains a set DIS of *discovered* nodes: these are nodes from $L_{k-1} \cup L_k \cup L_{k+1}$ whose labels the source learned in phase $k+1$. At the beginning of phase $k+1$ this set is empty. Each node from L_k appends the number k of its layer to all its messages.
Description of phase $k+1$:

Part 0. The aim of this part is the verification of whether layer L_k is empty or not. If $L_k = \emptyset$, then the radius of the network is $D = k-1$ and broadcasting was completed at the end of phase $k-1$, by Property 1 for k. In this case the source sends a stop message. Otherwise, the source sends a message requesting the start of Part 1. Here is a detailed description of Part 0.

- The source initiates broadcast of the message "start verification in step t," by consecutive use of Procedures SEND(j), for $j = 1, \ldots, k$, according to Property 2 for k. Step t is calculated to guarantee reception of this message by all nodes in L_j, for $j = 1, \ldots, k$, i.e., to guarantee completion of all Procedures SEND(j).
- Verification of whether layer L_k is empty starts in step t.
  - Using Procedure DETECT$(k-1, L_k)$, nodes from L_{k-1} detect if they have neighbors in L_k.
  - Let A_{k-1} be the set of nodes from L_{k-1} which detected neighbors in L_k. Using Procedure DETECT$(k-2, A_{k-1})$, nodes from L_{k-2} detect if they have neighbors in A_{k-1}. This process continues with sets $A_i \subseteq L_i$, until the source detects if it has neighbors in A_1.
- If the source does not have neighbors in A_1 (i.e., A_1 is empty, which means that L_k is empty as well), then it initiates broadcast of the message "stop in step t_1" (for an appropriately calculated t_1), by consecutive use of Procedures SEND(j), for $j = 1, \ldots, k-1$, according to Property 2 for k. Otherwise, a message requesting the start of Part 1 in an appropriately calculated step is sent similarly as above (to all layers L_j for $j = 1, \ldots, k$).

Part 1. The aim of this part is selection of consecutive nodes from L_k which

have at least d'_{k+1} undiscovered neighbors. This is done as follows, using a similar cascade of Procedures DETECT, as in Part 0.

- Let B_k be the set of nodes from L_k which have at least d'_{k+1} undiscovered neighbors. Using Procedure DETECT$(k-1, B_k)$, nodes from L_{k-1} detect if they have neighbors in B_k.
- Let B_{k-1} be the set of nodes from L_{k-1} which detected neighbors in B_k. Using Procedure DETECT$(k-2, B_{k-1})$, nodes from L_{k-2} detect if they have neighbors in B_{k-1}. This process continues with sets $B_i \subseteq L_i$, until the source detects if it has neighbors in B_1.
- If the source does not have neighbors in B_1 (i.e., B_1 is empty), then it initiates broadcast of the message "go to Part 2 in step t_2" (for an appropriately calculated t_2), by consecutive use of Procedures SEND(j) for $j = 1, \ldots, k$, according to Property 2 for k. Otherwise the source selects one node from B_1, using Procedure SELECT$(0, 0, B_1)$.
- The selected node v performs SELECT$(1, v, B_2)$ to select one of its neighbors in B_2. This is continued until one node w in B_k is selected.
- Node w broadcasts a message (containing the source message, the label w, and labels of neighbors of w). All these neighbors get the source message. Moreover, broadcast is propagated along the path containing selected nodes from consecutive sets B_i. The source discovers neighbors of w and possibly w itself, updates the set DIS, and propagates this information to all nodes in layers $L_1, \ldots, L_k$, using SEND procedures.

This selection process continues until all nodes in L_k have less than d'_{k+1} undiscovered neighbors.

Part 2. Let X be the set of all undiscovered nodes in L_k, and Y the set of all undiscovered nodes in $L_{k-1} \cup L_k \cup L_{k+1}$ which have at most d_{k+1} neighbors in X. A node can tell if it is in X, since in view of Property 1 it knows if it is in L_k, and after Part 1 of phase $k+1$ it knows whether it is in set DIS.

Apply the protocol from Theorem 4.2 (for $d = d_{k+1}$ and for the source message) to the subgraph of G induced by $X \cup Y$. At the end of Part 2, all nodes from Y got the source message.

Part 3. Consider the subgraph G of the radio network induced by the set of nodes V consisting of all undiscovered nodes in $L_{k-1} \cup L_k$, and of all undiscovered nodes in L_{k+1} which got the source message in Part 2. Each node knows if it is in V, in view of Part 2, of the knowledge of DIS gotten at the end of Part 1, and of Property 1.

Apply the protocol from Theorem 4.2 to the graph G (for $d = d'_{k+1}$). The message transmitted by each node contains its label and the source message. At the end of Part 3, all undiscovered nodes in L_k, which have at most d'_{k+1} neighbors in G, know which of their neighbors in G got the source message.

Part 4. All undiscovered nodes from L_k check if all their undiscovered neighbors got the source message. Consider the set Z of undiscovered nodes from L_k which did not get a message in Part 3 from some of their undiscovered neighbors. As in Part 1, we do the following:

- Procedures DETECT and SELECT are used to select one node in Z,
- this node transmits (alone) the source message and the set of all its neighbors in L_{k+1},
- this message is propagated to the source,
- the source updates the set DIS of discovered nodes (now DIS includes the selected node from Z and all of its neighbors) and propagates DIS to layer L_k.

After each selection, at least one currently undiscovered node in L_{k+1} gets the source message. This process continues until all nodes in L_k know that all their neighbors received the source message. □

THEOREM 4.5. *Algorithm Sublinear-Broadcast completes broadcasting in arbitrary radio networks.*

Proof. It is enough to prove that Properties 1 and 2 are satisfied after each phase $k > 1$. First we prove them for $k = 2$, i.e., upon completion of Algorithm $\mathcal{A}_2$.

Property 1. By correctness of Algorithm $\mathcal{A}_2$, all nodes in L_1 and L_2 get the source message. All nodes in L_1 know that they are in L_1 because they got the message directly from the source. All nodes from L_2 got the message from some neighbor in L_1, by the description of Algorithm $\mathcal{A}_2$. On the other hand, they know that they are not in L_1, so they deduce that they are in L_2.

Property 2. Upon completion of Algorithm $\mathcal{A}_2$, all nodes in L_2 either are discovered or have at most d_2 neighbors in L_2. Procedure SEND(2) can be executed as follows. Broadcasting assuming knowledge of topology is executed in the graph containing all nodes from L_1 and all discovered nodes from L_2. This is done according to the protocol from Theorem 4.3 in time $\alpha_1 \cdot \log^2 r$. Broadcasting to the remaining nodes is done using the protocol from Theorem 4.1 in time $\alpha_2 \cdot (d_2 \log r)$. The property follows for $\alpha = \max(\alpha_1, \alpha_2)$.

Now assume that Properties 1 and 2 hold after phase k of Algorithm Sublinear-Broadcast. We have to prove that they hold after phase $k + 1$.

Property 1. Every node v in L_{k+1} gets a message from a neighbor w in L_k in phase $k + 1$. This is proved as follows. If v did not get a message until the end of Part 3, then all neighbors of v in L_k know that v did not get a message. Then at least one neighbor of v from L_k is selected in Part 4 and v gets a message from it. Node v learns that neighbor w is in L_k because w knows this by Property 1 after phase k and attaches this information to its message. Node v also knows that itself is not in L_i for $i \leq k$, so it deduces that it is in L_{k+1}.

Property 2. Procedure SEND$(k + 1)$ is executed similarly as SEND(2) (described above), by replacing L_1 by L_k, L_2 by L_{k+1}, and d_2 by d_{k+1}. □

Our next result estimates time complexity of Algorithm Sublinear-Broadcast for networks of small radius.

THEOREM 4.6. *Algorithm Sublinear-Broadcast completes broadcasting in time $o(n)$, for all n-node radio networks of radius $o(\log \log n)$.*

Proof. Fix a phase $k > 2$ of the algorithm. We estimate time complexity of each of its five parts separately.

Part 0. All Procedures SEND take a total of at most $2\alpha \cdot (\sum_{j<k}(d_j + \log r) \log r)$ steps. All Procedures DETECT take a total of at most $k - 1 + \alpha \cdot (\sum_{j<k-1}(d_j + \log r) \log r)$ steps. Hence the number of steps in the entire Part 0 is at most $\mathcal{O}(\sum_{j<k}(d_j + \log r) \log r) = \mathcal{O}(\sum_{j<k}(d_j + \log n) \log n)$.

Part 1. There can be at most (n/d'_k) selected nodes. We estimate the number of steps needed for each selection. All Procedures DETECT take a total of at most $k - 1 + \alpha \cdot (\sum_{j<k-1}(d_j + \log r) \log r)$ steps. All Procedures SELECT take a total of at most $\alpha(k - 1) \cdot \log r$ steps. Sending back a message to the source along a fixed path of selected nodes takes $k - 1$ steps. All Procedures SEND take a total of at most $\alpha \cdot (\sum_{j<k}(d_j + \log r) \log r)$ steps. Defining $\gamma = \alpha \cdot (\log r / \log n)^2$, we get the estimate

$$2\alpha \cdot \left(\sum_{j<k} (d_j + \log r) \log r \right) + \alpha(k-1) \cdot \log r + 2(k-1) \leq 4\gamma \cdot \left(\sum_{j<k} (d_j + \log n) \log n \right)$$

on the number of steps for each selection. Hence the number of steps in the entire Part 1 is at most $4\gamma \cdot ((n/d'_k) \cdot \sum_{j<k}(d_j + \log n)\log n) \in \mathcal{O}((n/d'_k) \cdot \sum_{j<k}(d_j + \log n)\log n)$.

Part 2. By Theorem 4.2, this part takes $\mathcal{O}(d_k^2 \log r) = \mathcal{O}(d_k^2 \log n)$ steps.

Part 3. By Theorem 4.2, this part takes $\mathcal{O}((d'_k)^2 \log r) = \mathcal{O}((d'_k)^2 \log n)$ steps.

Part 4. Every node in set Z has at most d'_k undiscovered neighbors (otherwise it would be selected in Part 1). Every undiscovered neighbor w of a node $v \in Z$, from which v did not get a message in Part 3, is not in the graph G (defined in Part 3), hence it does not have the source message yet. By the description of Part 2, node w has more than d_k neighbors in L_{k-1}. Hence at most nd'_k/d_k selections of nodes in Z will be performed. The number of steps for each node is $\mathcal{O}(\sum_{j<k}(d_j + \log n)\log n)$, similarly as in Part 1. This gives a total of $\mathcal{O}((nd'_k/d_k)\sum_{j<k}(d_j + \log n)\log n)$.

We now choose the following parameters: $d_1 = \lfloor \log r \rfloor$, $d'_{i+1} = d_i^2$, and $d_{i+1} = (d'_{i+1})^2$. This gives the following estimates of the numbers of steps in different parts of phase k:

Part 0: $\mathcal{O}(d_{k-1} \log n) \subseteq \mathcal{O}(d_k^2 \log n)$.

Part 1: $\mathcal{O}((n/d'_k) \cdot d_{k-1} \log n) \subseteq \mathcal{O}((n \log n)/d_{k-1})$.

Part 2: $\mathcal{O}(d_k^2 \log n)$.

Part 3: $\mathcal{O}((d'_k)^2 \log n) \subseteq \mathcal{O}(d_k^2 \log n)$.

Part 4: $\mathcal{O}((nd'_k/d_k) \cdot d_{k-1} \log n) \subseteq \mathcal{O}((n \log n)/d_{k-1})$.

Thus the entire phase k lasts $\mathcal{O}((n/d_{k-1}) \cdot \log n + d_k^2 \log n)$ steps.

The last phase of the algorithm run on a radio network of radius D is $D+2$ (in fact only Part 0 of phase $D+2$ is executed). The total time of phases $k = 3, \ldots, D+2$ is at most $\mathcal{O}(d_{D+2}^2 \log n + (n \log n)/d_2) \subseteq \mathcal{O}(\min(n, (\log n)^{4^{D+3}+1} + n/\log^2 n))$. Since $D \in o(\log \log n)$, we get that this time is $o(n)$. By Theorem 4.4 the time of the first two phases (occupied by execution of Algorithm $\mathcal{A}_2$) is $\mathcal{O}(n^{2/3} \log n)$, which is $o(n)$ as well. Hence the entire Algorithm Sublinear-Broadcast completes broadcasting in time $o(n)$. □

5. Lower bound. In this section we show that for every deterministic broadcasting algorithm $\mathcal{A}$, there exists a network $G_{\mathcal{A}}$ of radius 2, with at most $2n$ nodes, on which $\mathcal{A}$ requires $\Omega(\sqrt[4]{n})$ steps to complete broadcast. This network is chosen from the family $\mathcal{C}_n$ of networks defined as follows. Every graph $G \in \mathcal{C}_n$ (see Figure 5.1) consists of the source 0 and two layers $L_1 = \{1, \ldots, n\}$ and $L_2 = \{n+1, \ldots, n+q\}$, where q is the largest odd integer smaller than $\sqrt[4]{n}$. The source is adjacent to all nodes in L_1, and every node in L_1 is adjacent to exactly one node in L_2. These are the only edges in G. If G is fixed, we denote by V the set $\{0\} \cup L_1 \cup L_2$ of all nodes of G. For every node v, we denote by N_v the set of its neighbors.

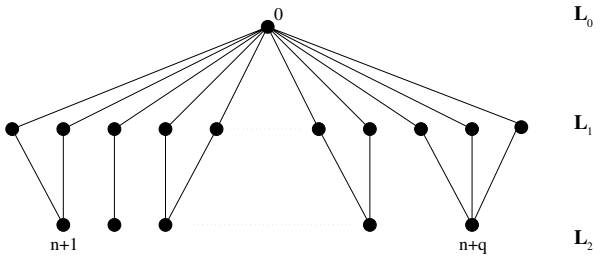


FIG. 5.1. *Network* $G \in \mathcal{C}_n$.

The idea of the proof is the following. We construct the network step by step, using consecutive steps of the fixed broadcasting algorithm $\mathcal{A}$, and *assuming* that particular nodes got particular messages in given steps. In order to express this, we use the notion of *abstract history* of a node, formally defined below. Intuitively, an abstract history of a node v at a given step k consists of a neighborhood of node v and of a sequence of messages received by this node until step k. Since the network is not yet constructed, neighborhoods of some nodes are not determined by step k, and consequently it is not yet known which abstract history will become the real one—the one given by algorithm $\mathcal{A}$ running on the final network. We can ensure that, if a given node had some abstract history up to a certain step, then it would behave in a given way (this is captured by the notion of abstract action function, defined below). Based on that we do the next step of the construction of the network (by determining neighborhoods of two nodes in layer L_2) and simultaneously define abstract histories of nodes in this step. These abstract histories are defined so as to prevent some nodes in layer L_2 of the network from getting any message for a long time. In particular, nodes of L_2 whose neighborhood is not yet determined have not gotten the source message so far.

When the construction is finished, we prove that if the algorithm $\mathcal{A}$ runs on the resulting network, then the real histories of all nodes are identical to the abstract (assumed) ones, and consequently some nodes of layer L_2 will indeed fail to receive the source message for $\Omega(\sqrt[4]{n})$ steps.

5.1. Construction. Fix a deterministic broadcasting algorithm $\mathcal{A}$. For this algorithm, running on any network, we define the following objects.

Histories and message format. H_k denotes the history of computation of algorithm $\mathcal{A}$ until the end of step k. This is the set $\{H_k(v) : v \in V\}$, where $H_k(v)$ is the history of computation at node v, until the end of step k. For any v and k, $H_k(v)$ is a sequence of (received) *messages* $(M_0(v), M_1(v), \ldots, M_k(v))$. Messages are defined inductively, as follows. $M_0(v)$ is either the triple $(\emptyset, \emptyset, \emptyset)$, called the *empty message*, or the triple $(0, N_0, \mathit{source\_message})$. $M_l(v)$ (for $l = 1, \ldots, k$) is the empty message if node v did not get any message in step l. Otherwise, it is a triple consisting of

- the label of node w from which node v received a message in step l,
- the set N_w,
- history $H_{l-1}(w)$.

Notice that we restrict attention to messages conveying the entire history of the transmitter. If a particular protocol requires transmitting specific information, the receiver can deduce this information from the received history, since programs of all nodes are the same. History $H_k(v)$ containing only empty messages is called the *empty history*.

Action function and sets of transmitters. Given algorithm $\mathcal{A}$, we denote by $\pi(v, N_v, H_{k-1}(v))$ the action of node v in step k, if its set of neighbors is N_v and its history until the end of step $k-1$ is $H_{k-1}(v)$. The values of the function π can be 1 or 0: if the value is 1, node v is sending the message $(v, N_v, H_{k-1}(v))$ in step k, otherwise it is receiving in step k. Under a fixed history H_{k-1}, we define the set of neighbors of v transmitting in step k as follows: $T_k(v) = \{w \in N_v : \pi(w, N_w, H_{k-1}(w)) = 1\}$.

We construct a network $G_{\mathcal{A}} \in \mathcal{C}_n$ on which $\mathcal{A}$ will work inefficiently. We first present the general overview of the construction and abstracts objects used in it. The construction is by induction on steps of the algorithm $\mathcal{A}$. The set of nodes $\{0\} \cup L_1 \cup L_2$, where $L_1 = \{1, \ldots, n\}$ and $L_2 = \{n+1, \ldots, n+q\}$, as well as all edges $\{(0, i) : i = 1, \ldots, n\}$ are given in the beginning. At each step of the induction some

edges between nodes from L_1 and L_2 are added. Since at each stage of the construction only a part of the network $G_{\mathcal{A}}$ is specified, the new edges are constructed using algorithm $\mathcal{A}$ and some *abstract history* $\hat{H}_k$ until this step. Abstract histories at each node are parametrized with a nonempty set A representing a possible neighborhood of v to be constructed in a later step.

Abstract objects. Abstract objects (messages, histories, action function, transmitters) are abstract versions of real objects, used in the construction because real ones do not exist until the network is completely defined. Let $v \in V$ and $A \subseteq V$. An abstract history $\hat{H}_k(v, A)$ of node v, assuming that its neighborhood is A, is defined as a sequence $(\hat{M}_0(v, A), \hat{M}_1(v, A), \ldots, \hat{M}_k(v, A))$ of *abstract messages*. $\hat{M}_0(v, A) = M_0(v)$, and $\hat{M}_l(v, A)$, for $l > 0$, either is the empty message or is of the format $(w, B, \hat{H}_{l-1}(w, B))$, for some $w \in V$ and $B \subseteq V$. We will construct the abstract history step by step, in parallel with the construction of network $G_{\mathcal{A}}$. Notice that, in general, abstract histories and abstract messages are not necessarily linked to any particular protocol.

We also define the *abstract action function* $\hat{\pi}(v, A, \hat{H}_{k-1}(v, A))$ as an extension of the action function π described above. For every v and A, if $\pi(v, A, \hat{H}_{k-1}(v, A))$ is defined, then $\hat{\pi}(v, A, \hat{H}_{k-1}(v, A)) = \pi(v, A, \hat{H}_{k-1}(v, A))$. Otherwise, $\hat{\pi}(v, A, \hat{H}_{k-1}(v, A)) = 0$.

We now define sets of *abstract transmitters*. First consider a node v with neighborhood N_v fixed at the end of step k of the construction, and assume that neighborhoods N_w of all nodes $w \in N_v$ are also fixed. Under a fixed abstract history $\hat{H}_{k-1}$, we define the set of abstract transmitters $\hat{T}_k(v, N_v) = \{w \in N_v : \hat{\pi}(w, N_w, \hat{H}_{k-1}(w, N_w)) = 1\}$.

Now define sets of abstract transmitters for nodes whose neighborhood is not yet fixed. Suppose that S_k is the set of all nodes j in L_2 for which the neighborhood N_j is not fixed until the end of step k of the construction. Suppose that R_k is the set of nodes in L_1 that do not belong to any fixed neighborhood at the end of step k, i.e., $R_k = L_1 \setminus \bigcup_{j \in L_2 \setminus S_k} N_j$. (Additionally, let $S_0 = L_2$ and $R_0 = L_1$.) For nodes of R_k and S_k, we define the sets of abstract transmitters in step k as follows:

- if $v \in R_k$, then for any $j \in S_k$, $\hat{T}_k(v, \{0, j\}) = \{0\}$ if $\hat{\pi}(0, L_1, \hat{H}_{k-1}(0, L_1)) = 1$, and $\hat{T}_k(v, \{0, j\}) = \emptyset$ otherwise;
- if $v \in S_k$ and $R \subseteq L_1$, then $\hat{T}_k(v, R) = \{i \in R : \hat{\pi}(i, \{0, v\}, \hat{H}_{k-1}(i, \{0, v\})) = 1\}$.

Sets R_k and S_k will be defined dynamically in a formal way, during step k of the construction. We will prove that these formal definitions correspond to the meaning intended above for R_k and S_k, by proving Property 1 of the invariant after step k.

We now describe the inductive construction of the graph $G_{\mathcal{A}}$. We begin by defining the abstract history $\hat{H}_0$. $\hat{H}_0(v, A) = (\hat{M}_0(v, A))$, for all nodes v and sets A, where $\hat{M}_0(v, A)$ is the empty message for all $v \notin L_1$, and $\hat{M}_0(v, A) = (0, L_1, source\_message)$, for all $v \in L_1$.

We now begin step 1 of the construction, on the basis of step 1 of the algorithm and of the abstract history $\hat{H}_0$ already defined. To this end we will need the function First-Step-Selection, formally described below. We want to choose an element j of the set S (corresponding to L_2), to which the largest number of elements of the set R (corresponding to L_1) would transmit, if they were neighbors of j. Then we determine neighbors of j in L_1. When the function determines j, it also determines its neighborhood, and it deletes j from S. Hence, if S was the set of nodes with undetermined neighborhood before applying the function, it will preserve this property after applying it. When the neighborhood of j is determined, then (since neighbors of j are in L_1 and nodes in L_1 have exactly one neighbor in L_2) we automatically

determine neighborhoods of neighbors of j. These neighbors are deleted from R, and hence R preserves the property of containing nodes with undetermined neighborhood, similar to S.

FUNCTION FIRST-STEP-SELECTION(R, S).

- Choose some node $j \in S$ such that the size of $X = \hat{T}_1(j, R)$ is maximal, and put two nodes from X to N_j (or one if X has one element, or nothing if X is empty); then remove these nodes from R. Remove j from S.
- Modify N_j as follows:
  if $N_j = \emptyset$, **then** put an arbitrary $i \in R$ to N_j and remove i from R;
  while there exists a node $i \in R$ such that $\hat{T}_1(v, R) = \{i\}$, for some $v \in S$
  do put i into N_j and remove i from R.
- Return (R, S, j, N_j).

Step 1 of the construction. The goal of step 1 of the construction is choosing two nodes j_1' and j_1 in L_2, together with their neighborhoods, in such a way that if some node from R_1 transmits in the first step of algorithm $\mathcal{A}$, then at least one other node from R_1 transmits as well. This is essential to guarantee the following property of abstract history $\hat{H}_1(0, L_1)$: no node from L_1 with yet undetermined neighborhood is heard by the source.

0. Initialize $R := L_1$ and $S := L_2$.
1. $(R, S, j_1', N_{j_1'}) :=$ FIRST-STEP-SELECTION(R, S);
   $R_1' := R$;
   $(R, S, j_1, N_{j_1}) :=$ FIRST-STEP-SELECTION(R, S);
   $R_1 := R$ and $S_1 := S$.
2. We construct the abstract history $\hat{H}_1$. Its definition corresponds to the definition of the "real" history, if neighborhoods are determined. Otherwise, the definition depends on the conditions on nodes and neighborhoods, the crucial case being the last item of the description below. History $\hat{H}_0$ is fixed; hence it is enough to define $\hat{M}_1(v, A)$, for all v, A. If $\hat{\pi}(v, A, \hat{H}_0(v, A)) = 1$, then $\hat{M}_1(v, A)$ is the empty message (transmitting nodes should not receive messages). Otherwise, $\hat{M}_1(v, A)$ is defined as follows:
   - Since nodes in $(L_1 \setminus R_1) \cup (L_2 \setminus S_1)$ have fixed neighborhoods, and also their neighbors have fixed neighborhoods, for each $v \in (L_1 \setminus R_1) \cup (L_2 \setminus S_1)$ we define $\hat{M}_1(v, N_v) = (w, N_w, \hat{H}_0(w, N_w))$, if $\hat{T}_1(v, N_v) = \{w\}$, and we define $\hat{M}_1(v, A)$ as the empty message in all other cases.
   - We define $\hat{M}_1(v, A)$ as the empty message, for all nodes $v \in S_1$ and all sets A.
   - We define $\hat{M}_1(0, L_1)$ as follows:
     – if $|\hat{T}_1(j_1', N_{j_1'}) \cup \hat{T}_1(j_1, N_{j_1})| \neq 1$, then $\hat{M}_1(0, L_1)$ is empty;
     – if $\hat{T}_1(j_1', N_{j_1'}) \cup \hat{T}_1(j_1, N_{j_1}) = \{i\}$, then $\hat{M}_1(0, L_1) = (i, N_i, \hat{H}_0(i, N_i))$. (Note that N_i is already defined at this point.)

     $\hat{M}_1(0, A)$ is defined as the empty message, for all $A \neq L_1$.
   - For every $v \in R_1$ and $j \in S_1$, if $\hat{T}_1(v, \{0, j\}) = \{0\}$, then $\hat{M}_1(v, \{0, j\}) = (0, L_1, \hat{H}_0(0, L_1))$, and $\hat{M}_1(v, A)$ is the empty message in all other cases.

This concludes the first step of the construction.

For any $k \geq 1$, the following invariant will be preserved after step k of the construction.

INVARIANT AFTER STEP k. The following objects are defined:
sets $S_l \subseteq L_2$ for $l = 0, 1, \ldots, k$;
sets $R_l \subseteq L_1$ for $l = 0, 1, \ldots, k$;

sets $R'_l \subseteq L_1$ for $l = 1, \ldots, k$;
nodes j'_l, j_l such that $S_{l-1} \setminus S_l = \{j'_l, j_l\}$ and $R_{l-1} \setminus R_l = N_{j'_l} \cup N_{j_l}$ for $l = 1, \ldots, k$.
The following properties hold:

1. Neighborhoods of nodes in $\{0\} \cup (L_1 \setminus R_k) \cup (L_2 \setminus S_k)$ are defined.
2. Histories $\hat{H}_k(v, A)$ are defined, for all nodes v and all sets A.
3. For all sets A, histories $\hat{H}_k(j, A)$ are empty for all $j \in S_k$, and histories $\hat{H}_{k-1}(j, A)$ are empty for all $j \in S_{k-1} \setminus S_k$.
4. For all nodes $j \in S_k \cup \{j_k\}$ and steps $l \leq k$, we have $|\hat{T}_l(j, R'_k)| \neq 1$.
5. For all nodes $j \in S_k$ and steps $l \leq k$, we have $|\hat{T}_l(j, R_k)| \neq 1$.
6. For all nodes $j \in S_{k-1} \setminus S_k$ and steps $l < k$, we have $|\hat{T}_l(j, N_j)| \neq 1$.

We now begin step $k+1$ of the construction, on the basis of step $k+1$ of the algorithm and of the invariant after step k. We will need a function similar to Function First-Step-Selection. Its aim is to choose $j \in S$ with the property as before (see the comment preceding the description of Function First-Step-Selection). This is done in the first item of the formal description given below. We also need to modify the neighborhood of j, so that choices (and elimination) of such nodes in previous steps of the construction do not yield a single transmitter to nodes with yet undetermined neighborhood. This is required in order to preserve Properties 4, 5, and 6 of the invariant. Modification of the neighborhood is done in the second item of the following formal description.

Function $(k+1)$st-Step-Selection(R, S).

- Choose some node $j \in S$ such that the size of $X = \hat{T}_{k+1}(j, R)$ is maximal and put two nodes from X to N_j (or one if X has one element, or nothing if X is empty), then remove these nodes from R. Remove j from S.
- Modify N_j as follows:
  if $N_j = \emptyset$, **then** put an arbitrary $i \in R$ to N_j and remove i from R;
  - **set** `stop` $:= 0$,
  - **while** `stop` $= 0$ **do**
    - **set** `stop` $:= 1$,
    - **while** there exists a node $i \in R$ such that $\hat{T}_l(j', R) = \{i\}$, for some $l \leq k+1$, $j' \in S$
      do put i into N_j and remove i from R,
    - **while** there exists a node $i \in N_j$ such that $\hat{T}_l(j, N_j) = \{i\}$, for some $l \leq k$
      do find another node $i' \in \hat{T}_l(j, R)$ (if it exists), put i' into N_j, and remove i' from R, **set** `stop` $:= 0$;
- Return (R, S, j, N_j).

Step $(k+1)$ of the construction (see Figure 5.2). The goal of step $(k+1)$ of the construction is choosing two nodes, $j'_{k+1}, j_{k+1} \in S_k$ (in L_2), together with their neighborhoods (included in R_k), and defining abstract history $\hat{H}_{k+1}$, so as to satisfy the invariant after step $(k+1)$ of the construction. Note that we do not initialize variables R and S because their values have been fixed after step k of the construction; indeed, at the beginning of step $k+1$, we have $R = R_k$ and $S = S_k$.

1. $(R, S, j'_{k+1}, N_{j'_{k+1}}) := (k+1)$st-Step-Selection$(R, S)$;
   $R'_{k+1} := R$;
   $(R, S, j_{k+1}, N_{j_{k+1}}) := (k+1)$st-Step-Selection$(R, S)$;
   $R_{k+1} := R$ and $S_{k+1} := S$.
2. We construct the abstract history $\hat{H}_{k+1}$. Its definition corresponds to the definition of the "real" history, if neighborhoods are determined. Otherwise,

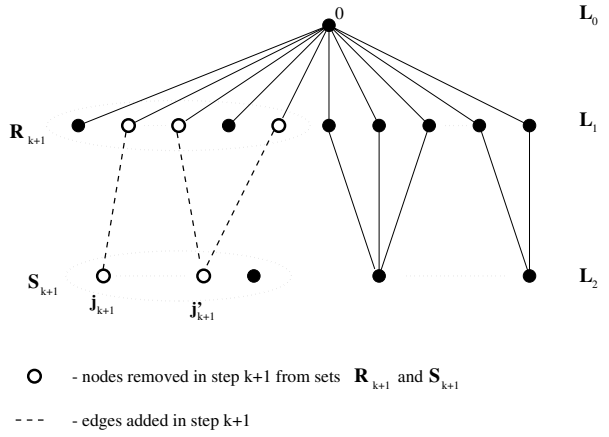


FIG. 5.2. *Step $k+1$ of the construction of $G_{\mathcal{A}} \in \mathcal{C}_n$.*

the definition depends on the conditions on nodes and neighborhoods, the crucial case being the last item of the description below. History $\hat{H}_k$ is fixed; hence it is enough to define $\hat{M}_{k+1}(v, A)$, for all v, A. If $\hat{\pi}(v, A, \hat{H}_k(v, A)) = 1$, then $\hat{M}_{k+1}(v, A)$ is the empty message (transmitting nodes should not receive messages). Otherwise, $\hat{M}_{k+1}(v, A)$ is defined as follows:

- Since nodes in $(L_1 \setminus R_{k+1}) \cup (L_2 \setminus S_{k+1})$ have fixed neighborhoods, and also their neighbors have fixed neighborhoods, for each $v \in (L_1 \setminus R_{k+1}) \cup (L_2 \setminus S_{k+1})$ we define $\hat{M}_{k+1}(v, N_v) = (w, N_w, \hat{H}_k(w, N_w))$, if $\hat{T}_{k+1}(v, N_v) = \{w\}$, and we define $\hat{M}_{k+1}(v, A)$ as the empty message in all other cases.
- We define $\hat{M}_{k+1}(v, A)$ as the empty message, for all nodes $v \in S_{k+1}$ and all sets A.
- We define $\hat{M}_{k+1}(0, L_1)$ as follows:
  - if $|\bigcup_{j \in L_2 \setminus S_{k+1}} \hat{T}_{k+1}(j, N_j)| \neq 1$, then $\hat{M}_{k+1}(0, L_1)$ is empty;
  - if $\bigcup_{j \in L_2 \setminus S_{k+1}} \hat{T}_{k+1}(j, N_j) = \{i\}$, then $\hat{M}_{k+1}(0, L_1) = (i, N_i, \hat{H}_k(i, N_i))$. (Note that N_i is already defined at this point.)

  $\hat{M}_{k+1}(0, A)$ is defined as the empty message, for all $A \neq L_1$.
- For every $v \in R_{k+1}$ and $j \in S_{k+1}$, if $\hat{T}_{k+1}(v, \{0, j\}) = \{0\}$, then $\hat{M}_{k+1}(v, \{0, j\}) = (0, L_1, \hat{H}_k(0, L_1))$, and $\hat{M}_{k+1}(v, A)$ is the empty message in all other cases.

5.2. Analysis. We first show that the invariant after step k of the construction holds if sets R_k, S_k are nonempty. This guarantees the correctness of the construction until one of these sets becomes empty, i.e., until all nodes either of L_1 or of L_2 have determined neighborhoods. Next we show that sets R_k, S_k are nonempty for $k \leq \frac{q-1}{2}$, where q is the largest odd integer smaller than $\sqrt[4]{n}$. This implies that the construction is correct until step $k = \frac{q-1}{2}$. Then we show how to finish the construction of network $G_{\mathcal{A}}$. Finally, we prove that histories determined by algorithm $\mathcal{A}$ running on network $G_{\mathcal{A}}$ are identical to the previously constructed abstract histories. In view of the invariant after step $k = \frac{q-1}{2} \in \Omega(\sqrt[4]{n})$ of the construction, this implies the desired lower bound on broadcasting time.

LEMMA 5.1. *The invariant after step k is preserved, for all $k \geq 1$ such that S_k and R_k are nonempty.*

Proof. The validity of the invariant after step 1 follows from the exit conditions in the first and second executions of function FIRST-STEP-SELECTION(R, S), in Part 1 of step 1 of the construction.

Assume that the invariant holds after step k and that S_{k+1} and R_{k+1} are nonempty. We prove that it holds after step $k+1$.

All required objects are defined by the construction in step $k+1$, using nonemptiness of S_{k+1} and R_{k+1}. It remains to prove the six properties.

1. This follows from Property 1 of the invariant after step k and from the construction of j'_{k+1}, j_{k+1}, and of their neighborhoods during Part 1 of the construction in step $k+1$.
2. $\hat{H}_{k+1}$ was defined in Part 2 of step $(k+1)$ of the construction.
3. The fact that $\hat{H}_{k+1}(v, A)$ is empty for all nodes $v \in S_{k+1}$ and all sets A follows from the assumption in Part 2 of step $(k+1)$ of the construction. The fact that $\hat{H}_k(j'_{k+1}, A)$ and $\hat{H}_k(j_{k+1}, A)$ are empty (j'_{k+1}, j_{k+1} are the only elements of $S_k \setminus S_{k+1}$) follows from Property 3 of the invariant after step k.
4. We prove that, for all nodes $j \in S_k \cup \{j_k\}$ and steps $l \leq k+1$, we have $|\hat{T}_l(j, R'_{k+1})| \neq 1$. This follows from the exit conditions of the external and of the first internal loop in function $(k+1)$ST-STEP-SELECTION(R, S) (more precisely, in the first execution of this function in Part 1 of step $(k+1)$ of the construction). The execution of the external loop ends if and only if the value of `stop` becomes 1, which means that the condition in the second internal loop is always false in the last turn of the external loop. Hence the condition of the first internal loop must be false at the end of the last turn of the external loop. This implies $|\hat{T}_l(j, R'_{k+1})| \neq 1$, for all nodes $j \in S_{k+1}$ and steps $l \leq k+1$.
5. We prove that for all nodes $j \in S_{k+1}$ and steps $l \leq k+1$ we have $|\hat{T}_l(j, R_{k+1})| \neq 1$. This follows by an argument similar as above, applied to function $(k+1)$ST-STEP-SELECTION(R, S) (now we refer to the second execution of this function in Part 1 of step $(k+1)$ of the construction).
6. The property $|\hat{T}_l(j, N_j)| \neq 1$, for all nodes $j \in S_k \setminus S_{k+1}$ and steps $l < k+1$, follows from the exit condition of the second internal loop in the first and second executions of function $(k+1)$ST-STEP-SELECTION(R, S), in Part 1 of step $(k+1)$ of the construction. Observe that the existence of i' in the second internal loop follows from Property 5 of the invariant after step k and from (the just proved) Property 4 of the invariant after step $k+1$. ∎

LEMMA 5.2. *The inductive construction of the network can be carried out for at least* $\frac{q-1}{2}$ *steps, where* q *is the largest odd integer smaller than* $\sqrt[4]{n}$.

Proof. Let $k \leq (q-1)/2$. Sets S_k are decreased by two nodes during one step; hence $S_k \neq \emptyset$, since $|S_0| = q$.

Claim. Sets R_k are decreased by at most $2q^3$ nodes during one step, at most q^3 for each of the chosen nodes j'_k, j_k.

This can be computed by analyzing loops in both executions of function kTH-STEP-SELECTION(R, S) in Part 1 of step k of the construction. Every turn of each of the internal loops increases the neighborhood $N_{j'_k}$ (resp., N_{j_k}) by at most one element and consequently decreases R by at most one element.

Consider the first execution. During the first internal loop, at most $kq \leq q^2/2$ nodes can be added to $N_{j'_k}$, since each action makes one set $\hat{T}_l(j, R)$ empty, where $l \leq k$, and $n+1 \leq j \leq n+q$. (Since we analyze subsequent executions of the loop in the function, symbol R in the expressions containing R corresponds to the current value of this variable, which changes dynamically. Hence values of these expressions

may also change dynamically.)

During the second internal loop, at most $k-1 \leq q/2$ nodes can be added to $N_{j'_k}$, since each action makes one set $\hat{T}_l(j'_k, N_{j'_k})$ of size at least 2, where $l \leq k-1$. There may be at most $k-1 \leq q/2$ executions of the external loop, since every execution of the external loop increases the size $|\hat{T}_l(j'_k, N_{j'_k})|$ to at least 2 for some $l \leq k-1$ while performing the second internal loop. When $|\hat{T}_l(j'_k, N_{j'_k})| \neq 1$, for all $l \leq k-1$, the execution of the external loop stops. Hence $N_{j'_k}$ is bounded by $(q^2/2+q/2)\cdot(q/2) \leq q^3$, and consequently R is decreased at the rate of at most q^3 nodes per step. The same is true for the second execution. Hence R_k is smaller than R_{k-1} by at most $2q^3$ nodes, which concludes the proof of the claim.

Since $q < \sqrt[4]{n}$, we have $R_k \neq \emptyset$ for all $k \leq (q-1)/2$. It follows that the construction can be carried out for $(q-1)/2$ steps. □

Using Lemma 5.2, the construction of the network $G_{\mathcal{A}}$ can be now concluded as follows. All nodes in $R_{(q-1)/2}$ are made adjacent to the only node in $S_{(q-1)/2}$. It follows from the construction that $G_{\mathcal{A}}$ belongs to the class $\mathcal{C}_n$ defined in the beginning of this section.

The histories $\hat{H}_k$ in consecutive steps of the construction were abstract histories defined in order to continue the construction in subsequent steps. The next lemma shows that the actual histories $H_k(v)$ of all nodes v of network $G_{\mathcal{A}}$ obtained by running algorithm $\mathcal{A}$ on this network, are identical to abstract histories $\hat{H}_k(v, N_v)$.

LEMMA 5.3. *Let $k \leq (q-1)/2$ be a step of the execution of algorithm $\mathcal{A}$ on network $G_{\mathcal{A}}$. Then $H_k(v) = \hat{H}_k(v, N_v)$, for all nodes v of network $G_{\mathcal{A}}$.*

Proof. In the first step of the algorithm execution, the source transmits and all nodes in L_1 receive the message. Nodes in L_2 receive nothing. Hence $H_0(v) = \hat{H}_0(v, N_v)$ by definition of $\hat{H}_0$.

Note that the definition of abstract history in step 1 of the construction is the same as that in step $k+1$, taken for $k=0$. Hence it is not necessary to separately analyze step 1, and we can proceed with the argument by induction, for an arbitrary k.

Assume by induction that $H_k(v) = \hat{H}_k(v, N_v)$, where $k < (q-1)/2$. We prove $H_{k+1}(v) = \hat{H}_{k+1}(v, N_v)$ by showing $M_{k+1}(v) = \hat{M}_{k+1}(v, N_v)$. (Since $k+1 \leq (q-1)/2$, the abstract history $\hat{H}_{k+1}$ is well defined, in view of Lemma 5.2.) Observe that, since $\hat{\pi}$ is an extension of π and $H_k(v) = \hat{H}_k(v, N_v)$, we have $\hat{\pi}(v, N_v, \hat{H}_k(v, N_v)) = \pi(v, N_v, H_k(v))$. Hence, if $\hat{\pi}(v, N_v, \hat{H}_k(v, N_v)) = 1$, then v acts as a transmitter in step $k+1$, and hence both $M_{k+1}(v)$ and $\hat{M}_{k+1}(v, N_v)$ are empty messages. Thus we assume in the following that $\hat{\pi}(v, N_v, \hat{H}_k(v, N_v)) = 0$, i.e., that v acts as a receiver.

Case 1. $v \in (L_1 \setminus R_{k+1}) \cup (L_2 \setminus S_{k+1})$.

By Property 1 of the invariant after step $k+1$ of the construction, v has a fixed neighborhood and all of its neighbors w have fixed neighborhoods. Since $\hat{H}_k(w, N_w) = H_k(w)$, we get $M_{k+1}(v) = \hat{M}_{k+1}(v, N_v)$.

Case 2. $v \in S_{k+1}$.

$\hat{M}_{k+1}(v, N_v)$ is the empty message by Property 3 of the construction invariant after step $k+1$. Let k' be the step in which the neighborhood N_v was constructed. Since $v \in S_{k+1}$, we have $k' > k+1$. By Property 6 of the invariant after step k' of the construction, $|\hat{T}_l(v, N_v)| \neq 1$, for all steps $l < k'$. In particular, $|\hat{T}_{k+1}(v, N_v)| \neq 1$. Since $\hat{H}_k(w, N_w) = H_k(w)$ for all $w \in N_v$, we have $T_{k+1}(v) = \hat{T}_{k+1}(v, N_v)$, and hence $T_{k+1}(v)$ is not a singleton. It follows that $M_{k+1}(v)$ is the empty message.

Case 3. $v = 0$.

If $j \in L_2 \setminus S_{k+1}$, then $\hat{T}_{k+1}(j, N_j) = T_{k+1}(j)$, since $\hat{H}_k(w, N_w) = H_k(w)$, for all $w \in N_j$. Hence $\bigcup_{j \in L_2 \setminus S_{k+1}} \hat{T}_{k+1}(j, N_j) = \bigcup_{j \in L_2 \setminus S_{k+1}} T_{k+1}(j)$. We consider three cases.

If $|\bigcup_{j \in L_2 \setminus S_{k+1}} \hat{T}_{k+1}(j, N_j)| > 1$, then $\hat{M}_{k+1}(0, L_1)$ is empty. In this case, $|T_{k+1}(0)| = |\bigcup_{j \in L_2} T_{k+1}(j)| > 1$ and hence $M_{k+1}(0)$ is the empty message.

If $\bigcup_{j \in L_2 \setminus S_{k+1}} \hat{T}_{k+1}(j, N_j) = \{i\}$, then $\hat{M}_{k+1}(0, L_1) = (i, N_i, \hat{H}_k(i, N_i))$. In this case, $\bigcup_{j \in L_2 \setminus S_{k+1}} T_{k+1}(j) = \{i\}$. By construction of j_{k+1}, we have $\hat{T}_{k+1}(j_{k+1}, R_{k+1}) = \emptyset$, and hence $\hat{T}_{k+1}(j, R_{k+1}) = \emptyset$, for all $j \in S_{k+1}$. Consequently, $T_{k+1}(j) = \hat{T}_{k+1}(j, N_j) \subseteq \hat{T}_{k+1}(j, R_{k+1}) = \emptyset$. It follows that $M_{k+1}(0) = (i, N_i, H_k(i))$. In view of $H_k(i) = \hat{H}_k(i, N_i)$, we get $M_{k+1}(0) = \hat{M}_{k+1}(0, L_1)$.

If $\bigcup_{j \in L_2 \setminus S_{k+1}} \hat{T}_{k+1}(j, N_j) = \emptyset$, then $\hat{M}_{k+1}(0, L_1)$ is the empty message. In this case, $\bigcup_{j \in L_2 \setminus S_{k+1}} T_{k+1}(j) = \emptyset$. The same reasoning as above gives $\hat{T}_{k+1}(j, R_{k+1}) = \emptyset$ for all $j \in S_{k+1}$. Consequently, $M_{k+1}(0)$ is the empty message.

Case 4. $v \in R_{k+1}$.

Since for all $j \in S_{k+1}$, $\hat{H}_{k+1}(j, N_j) = H_{k+1}(j)$ is the empty history, it follows that each node $v \in R_{k+1}$ can receive a message in step $k+1$ of $\mathcal{A}$ only from node 0, if this node transmits. If node 0 transmits in step $k+1$ of $\mathcal{A}$, then v receives the message $M_{k+1}(v) = (0, L_1, H_k(0))$. Since $H_k(0) = \hat{H}_k(0, L_1)$, by definition we have $\hat{T}_{k+1}(v, N_v) = \{0\}$. By construction of message $\hat{M}_{k+1}(v, N_v)$ we get $\hat{M}_{k+1}(v, N_v) = (0, L_1, \hat{H}_k(0, L_1))$. Since $H_k(0) = \hat{H}_k(0, L_1)$, we have $\hat{M}_{k+1}(v, N_v) = M_{k+1}(v)$. If node 0 does not transmit in step $k+1$ of $\mathcal{A}$, then $M_{k+1}(v)$ is empty. Since $H_k(0) = \hat{H}_k(0, L_1)$, by definition we have $\hat{T}_{k+1}(v, N_v) = \emptyset$. By construction, $\hat{M}_{k+1}(v, N_v)$ is the empty message. □

THEOREM 5.4. *For any deterministic broadcasting algorithm $\mathcal{A}$, there exists a network $G_{\mathcal{A}}$ of radius 2, with at most $2n$ nodes, for which this algorithm requires time $\Omega(\sqrt[4]{n})$.*

Proof. Network $G_{\mathcal{A}}$ constructed above has $n + 1 + q \leq 2n$ nodes, since q is the largest odd integer smaller than $\sqrt[4]{n}$. It has radius 2 by construction. Let $k = (q-1)/2$. By Lemma 5.2, S_k is nonempty. By Lemma 5.1 and Property 3 of the invariant after step k, histories $\hat{H}_k(j, N_j)$ are empty for all $j \in S_k$. By Lemma 5.3, histories $H_k(j)$ are empty for all $j \in S_k$. Hence no node in S_k receives the source message by step k of algorithm $\mathcal{A}$. It follows that algorithm $\mathcal{A}$ requires time $\Omega(\sqrt[4]{n})$ to broadcast on network $G_{\mathcal{A}}$. □

Using the above technique we can prove the following more general result.

COROLLARY 5.5. *For any deterministic broadcasting algorithm $\mathcal{A}$ and any parameters $D \leq n$, there exists an n-node network of radius D, for which this algorithm requires time $\Omega(\sqrt[4]{nD^3})$.*

6. Conclusion. In this paper we studied deterministic broadcasting time in radio networks whose nodes know only their immediate neighborhood. We presented an algorithm for broadcasting in sublinear time in all networks of radius $o(\log \log n)$ and we proved a lower bound $\Omega(\sqrt[4]{n})$ on broadcasting time even in networks of radius 2. In view of the randomized algorithm from [3] running in expected time $\mathcal{O}(D \log n + \log^2 n)$ on all n-node graphs of diameter D, our lower bound proves an exponential gap between time of deterministic and randomized broadcasting in radio networks.

The main problem that remains open is the following. Is there a deterministic broadcasting algorithm running in sublinear time on all networks with sublinear radius, if nodes know only their immediate neighborhood? If complete knowledge of the network is available, the positive answer follows from [18].

REFERENCES

[1] N. Alon, A. Bar-Noy, N. Linial, and D. Peleg, *A lower bound for radio broadcast*, J. Comput. System Sci., 43 (1991), pp. 290–298.

[2] B. Awerbuch, *A new distributed depth-first-search algorithm*, Inform. Process. Lett., 20 (1985), pp. 147–150.

[3] R. Bar-Yehuda, O. Goldreich, and A. Itai, *On the time complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization*, J. Comput. System Sci., 45 (1992), pp. 104–126.

[4] R. Bar-Yehuda, O. Goldreich, and A. Itai, *Errata Regarding "On the time complexity of broadcast in radio networks: An exponential gap between determinism and randomization,"* http://www.wisdom.weizmann.ac.il/mathusers/oded/p_bgi.html, 2002.

[5] S. Basagni, D. Bruschi, and I. Chlamtac, *A mobility-transparent deterministic broadcast mechanism for ad hoc networks*, IEEE/ACM Trans. on Networking, 7 (1999), pp. 799–807.

[6] S. Basagni, A. D. Myers, and V. R. Syrotiuk, *Mobility-independent flooding for real-time multimedia applications in ad hoc networks*, in Proceedings of the IEEE Emerging Technologies Symposium on Wireless Communications & Systems, Richardson, TX, 1999.

[7] D. Bruschi and M. Del Pinto, *Lower bounds for the broadcast problem in mobile radio networks*, Distr. Comp., 10 (1997), pp. 129–135.

[8] I. Chlamtac and A. Faragó, *Making transmission schedule immune to topology changes in multi-hop packet radio networks*, IEEE/ACM Trans. on Networking, 2 (1994), pp. 23–29.

[9] I. Chlamtac, A. Faragó, and H. Zhang, *Time-spread multiple access (TSMA) protocols for multihop mobile radio networks*, IEEE/ACM Trans. on Networking, 5 (1997), pp. 804–812.

[10] I. Chlamtac and O. Weinstein, *The wave expansion approach to broadcasting in multihop radio networks*, IEEE Trans. on Communications, 39 (1991), pp. 426–433.

[11] B. S. Chlebus, L. Gasieniec, A. Gibbons, A. Pelc, and W. Rytter, *Deterministic broadcasting in unknown radio networks*, Distributed Computing, 15 (2002), pp. 27–38.

[12] B. S. Chlebus, L. Gąsieniec, A. Östlin, and J. M. Robson, *Deterministic radio broadcasting*, in Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP'2000), Lecture Notes in Comput. Sci. 1853, Springer-Verlag, Berlin, 2000, pp. 717–728.

[13] M. Chrobak, L. Gąsieniec, and W. Rytter, *Fast broadcasting and gossiping in radio networks*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Redondo Beach, CA, 2000, pp. 575–581.

[14] A. E. F. Clementi, A. Monti, and R. Silvestri, *Selective families, superimposed codes, and broadcasting on unknown radio networks*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2001, pp. 709–718.

[15] A. Czumaj and W. Rytter, *Broadcasting algorithms in radio networks with unknown topology*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, Cambridge, MA, 2003, pp. 492–501.

[16] G. De Marco and A. Pelc, *Faster broadcasting in unknown radio networks*, Inform. Process. Lett., 79 (2001), pp. 53–56.

[17] P. Erdős, P. Frankl, and Z. Füredi, *Families of finite sets in which no set is covered by the union of r others*, Israel J. Math., 51 (1985), pp. 79–89.

[18] I. Gaber and Y. Mansour, *Broadcast in radio networks*, in Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1995, pp. 577–585.

[19] F. K. Hwang, *The time complexity of deterministic broadcast radio networks*, Discrete Appl. Math., 60 (1995), pp. 219–222.

[20] P. Indyk, *Explicit constructions of selectors and related combinatorial structures, with applications*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2002, pp. 697–704.

[21] W. H. Kautz and R. R. C. Singleton, *Nonrandom binary superimposed codes*, IEEE Trans. on Inform. Theory, 10 (1964), pp. 363–377.

[22] D. Kowalski and A. Pelc, *Faster deterministic broadcasting in ad hoc radio networks*, in Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Lecture Notes in Comput. Sci. 2607, Springer-Verlag, Berlin, 2003, pp. 109–120.

[23] D. Kowalski and A. Pelc, *Broadcasting in undirected ad hoc radio networks*, in Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing, Boston, 2003, pp. 73–82.

[24] D. Kowalski and A. Pelc, *Time of radio broadcasting: Adaptiveness vs. obliviousness and randomization vs. determinism*, in Proceedings of the 10th Colloquium on Structural Information and Communication Complexity, Umea, Sweden, 2003, pp. 195–210.

[25] E. Kushilevitz and Y. Mansour, *An $\Omega(D \log(N/D))$ lower bound for broadcast in radio networks*, SIAM J. Comput., 27 (1998), pp. 702–712.
[26] D. Peleg, *Deterministic Radio Broadcast with No Topological Knowledge*, manuscript, 2000.
[27] A. Ta-Shma, C. Umans, and D. Zuckerman, *Loss-less condensers, unbalanced expanders, and extractors*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 143–152.

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 892–922

THE PARAMETERIZED COMPLEXITY OF COUNTING PROBLEMS*

JÖRG FLUM† AND MARTIN GROHE‡

Abstract. We develop a parameterized complexity theory for counting problems. As the basis of this theory, we introduce a hierarchy of parameterized counting complexity classes #W[t], for $t \geq 1$, that corresponds to Downey and Fellows's W-hierarchy [R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999] and we show that a few central W-completeness results for decision problems translate to #W-completeness results for the corresponding counting problems.

Counting complexity gets interesting with problems whose decision version is tractable, but whose counting version is hard. Our main result states that counting cycles and paths of length k in both directed and undirected graphs, parameterized by k, is #W[1]-complete. This makes it highly unlikely that these problems are fixed-parameter tractable, even though their decision versions are fixed-parameter tractable. More explicitly, our result shows that most likely there is no $f(k) \cdot n^c$-algorithm for counting cycles or paths of length k in a graph of size n for any computable function $f : \mathbb{N} \to \mathbb{N}$ and constant c, even though there is a $2^{O(k)} \cdot n^{2.376}$ algorithm for finding a cycle or path of length k [N. Alon, R. Yuster, and U. Zwick, *J. ACM*, 42 (1995), pp. 844–856].

Key words. counting complexity, parameterized complexity, paths and cycles, descriptive complexity

AMS subject classifications. 68Q15, 68Q19, 68Q25

DOI. 10.1137/S0097539703427203

1. Introduction.

Counting problems have been the source for some of the deepest and most fascinating results in computational complexity theory, ranging from Valiant's fundamental result [29] that counting perfect matchings of bipartite graphs is #P-complete over Toda's theorem [28] that the class $\mathrm{P}^{\#\mathrm{P}}$ contains the polynomial hierarchy to Jerrum, Sinclair, and Vigoda's [20] fully polynomial randomized approximation scheme for computing the number of perfect matchings of a bipartite graph. In this paper, we develop a basic parameterized complexity theory for counting problems.

Parameterized complexity theory provides a framework for a fine-grain complexity analysis of algorithmic problems that are intractable in general. In recent years, ideas from parameterized complexity theory have found their way into various areas of computer science, such as database theory [19, 24], artificial intelligence [18], and computational biology [6, 27]. Central to the theory is the notion of *fixed-parameter tractability*, which relaxes the classical notion of tractability, i.e., polynomial time computability, by admitting algorithms whose running time is exponential, but only in terms of some *parameter* of the problem instance that can be expected to be small in the typical applications. A good example is the evaluation of database queries: Usually, the size k of the query to be evaluated is very small compared to the size n of the database. An algorithm evaluating the query in time $O(2^k \cdot n)$ may therefore be acceptable, even quite good. On the other hand, an $\Omega(n^k)$ evaluation algorithm

*Received by the editors May 7, 2003; accepted for publication (in revised form) January 27, 2004; published electronically May 5, 2004.
http://www.siam.org/journals/sicomp/33-4/42720.html
†Institut für Mathematische Logik, Albert-Ludwigs-Universität Freiburg, Eckerstr. 1, 79104 Freiburg, Germany (Joerg.Flum@math.uni-freiburg.de).
‡Institut für Informatik, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany (grohe@informatik.hu-berlin.de).

usually cannot be considered feasible. Fixed-parameter tractability is based on this distinction: A parameterized problem is *fixed-parameter tractable* if there is a computable function f and a constant c such that the problem can be solved in time $f(k) \cdot n^c$, where n is the input size and k is the parameter value.

A standard example of a fixed-parameter tractable problem is the vertex cover problem parameterized by the size k of the vertex cover. It is quite easy to see that a vertex cover of size k of a graph of size n can be computed in time $O(2^k \cdot n)$ by a simple search tree algorithm based on the fact that at least one of the two endpoints of each edge must be contained in a vertex cover. (As a matter of fact, such an algorithm computes all minimum vertex covers of size at most k.) A standard example of a problem that does not seem to be fixed-parameter tractable is the clique problem, parameterized by the size of the clique. Indeed, all known algorithms for deciding whether a graph of size n has a clique of size k have a running time of $n^{\Omega(k)}$.

To give evidence that parameterized problems such as the clique problem are not fixed-parameter tractable, a theory of *parameterized intractability* has been developed (see [11, 12, 13]). It resulted in a rather unwieldy variety of parameterized complexity classes. The most important of these classes are the classes W[t], for $t \geq 1$, forming the so-called *W-hierarchy*. It is believed that W[1] strictly contains the class FPT of all fixed-parameter tractable problems and that the W-hierarchy is strict. Many natural parameterized problems fall into one of the classes of the W-hierarchy. For example, the parameterized clique problem is complete for the class W[1], and the parameterized dominating set problem is complete for the class W[2] (under suitable parameterized reductions).

So far, the parameterized complexity of counting problems has not been studied very systematically. A few tractability results are known: First, some fixed-parameter tractable decision problems have algorithms that can easily be adapted to the corresponding counting problems. An example is the vertex cover problem; since all minimum vertex covers of size at most k of a graph of size n can be computed in time $O(2^k \cdot n)$, a simple application of the inclusion-exclusion principle yields a fixed-parameter tractable counting algorithm for the vertex covers of size k. Similar counting algorithms are possible for other problems that have a fixed-parameter tractable algorithm based on the *method of bounded search tree* (see [13]). More interesting are results of Arnborg, Lagergren, and Seese [4], Courcelle, Makowsky, and Rotics [10], and Makowsky [22] stating that counting problems definable in monadic second-order logic (in various ways) are fixed-parameter tractable when parameterized by the tree-width of the input graph. For example, Arnborg, Lagergren, and Seese's result implies that counting the Hamiltonian cycles of a graph is fixed-parameter tractable when parameterized by the tree-width of the graph, and Makowsky's result implies that evaluating the Tutte polynomial is fixed-parameter tractable when parameterized by the tree-width of the graph. Courcelle, Makowsky, and Rotics [10] also proved similar results for graphs of bounded clique-width. Frick [16] showed that counting problems definable in first-order logic are fixed-parameter tractable on locally tree-decomposable graphs. For example, this implies that counting dominating sets of a planar graph is fixed-parameter tractable when parameterized by the size of the dominating sets.

We focus on the *intractability* of parameterized counting problems. We define classes #W[t], for $t \geq 1$, of parameterized counting problems that correspond to the classes of the W-hierarchy. Our first results show that a few central completeness results for the classes W[1] and W[2] translate to corresponding completeness results for the first two levels #W[1] and #W[2] of the #W-hierarchy. For example, we show

that counting cliques of size k is #W[1]-complete and counting dominating sets of size k is #W[2]-complete (both under *parsimonious parameterized reductions*). We then characterize the class #W[1] as the class of all counting problems that can be described in terms of numbers of accepting computations of certain nondeterministic programs. To give further evidence that the class #W[1] strictly contains the class of fixed-parameter tractable counting problems, we show that if this were not the case there would be a $2^{o(n)}$-algorithm counting the satisfying assignments of a 3-CNF-formula with n variables. This is the counting version of a result due to Abrahamson, Downey, and Fellows [1]. While these results are necessary to lay a solid foundation for the theory and not always easy to prove, by and large they do not give us remarkable new insights. The theory gets interesting with those counting problems that are harder than their decision versions.

Our main result states that *counting cycles and paths of length k in both directed and undirected graphs, parameterized by k, is* #W[1]*-complete under parameterized Turing reductions.* It is an immediate consequence of a theorem of Plehn and Voigt [26] that the decision versions of these problems are fixed-parameter tractable (but of course not in polynomial time, because if they were, the Hamiltonian path/cycle problem would be also). Alon, Yuster, and Zwick's [2] *color coding* technique provides algorithms for finding a path of length k in time $O(k! \cdot m)$ in a graph with m edges and for finding a cycle of length k in time $O(2^{O(k)} \cdot n^\omega)$ in a graph with n vertices, where $\omega < 2.376$ is the exponent of matrix multiplication. The hardness of the cycle counting problem in undirected graphs may be surprising in view of another algorithm due to Alon, Yuster, and Zwick [3] showing that cycles *up to length* 7 in an undirected graph can be counted in time $O(n^\omega)$. Our result implies that it is very unlikely that there is such an algorithm for counting cycles of arbitrary fixed length k.

The paper is organized as follows. After giving the necessary preliminaries in section 2, in section 3 we discuss fixed-parameter tractable counting problems. This section has the character of a short survey; apart from a few observations it contains no new results. In section 4, we introduce the #W-hierarchy and establish the basic completeness results. The hardness of counting cycles and paths is established in section 5. Definitions of all parameterized problems considered in this paper can be found in Appendix A.

We would like to point out that some of the results in section 4 have independently been obtained by others in two recent papers: McCartin [23] proves the #W[1]-completeness of clique and the #W[2]-completeness of dominating set. (Our proofs of these results are quite different from hers.) Furthermore, she shows that a number of further completeness results for parameterized decision problems translate to the corresponding counting problems. Arvind and Raman [5] also obtain the #W[1]-completeness of clique. Their main result is that the number of cycles or paths of length k can be approximated by a randomized fixed-parameter tractable algorithm. Indeed, they prove this not only for cycles and paths, but for arbitrary graphs of bounded tree-width. These results nicely complement our main result that exactly counting paths and cycles is hard.

2. Preliminaries.

2.1. Parameterized complexity theory. A *parameterized problem* is a set $P \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. If $(x, k) \in \Sigma^* \times \mathbb{N}$ is an instance of a parameterized problem, we refer to x as the *input* and to k as the *parameter*.

Definition 2.1. *A parameterized problem $P \subseteq \Sigma^* \times \mathbb{N}$ is* fixed-parameter tractable *if there is a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant $c \in \mathbb{N}$, and an algorithm*

that, given a pair $(x, k) \in \Sigma^* \times \mathbb{N}$*, decides if* $(x, k) \in P$ *in at most* $f(k) \cdot |x|^c$ *steps.*

We usually use k to denote the parameter and $n = |x|$ to denote the size of the input.

To illustrate our notation, let us give one example of a parameterized problem, the *parameterized vertex cover problem*, which is well known to be fixed-parameter tractable:

p-VC	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ has a vertex cover of size k.

From now on, we will give only brief definitions of the parameterized problems we consider in the main text; for exact definitions we refer the reader to Appendix A.

To define the classes of the W-hierarchy, we need a few notions from propositional logic. Formulas of propositional logic are built up from *propositional variables* $X_1, X_2, \ldots$ by taking conjunctions, disjunctions, and negations. The negation of a formula θ is denoted by $\neg\theta$. We distinguish between *small conjunctions*, denoted by $\wedge$, which are just conjunctions of two formulas, and *big conjunctions*, denoted by $\bigwedge$, which are conjunctions of arbitrary finite sets of formulas. Analogously, we distinguish between *small disjunctions*, denoted by $\vee$, and *big disjunctions*, denoted by $\bigvee$.

A formula is *small* if it contains only small conjunctions and small disjunctions. We define $\Gamma_0 = \Delta_0$ to be the class of all small formulas. For $t \geq 1$, we define Γ_t to be the class of all big conjunctions of formulas in Δ_{t-1} and we define Δ_t to be the class of all big disjunctions of formulas in Γ_{t-1}.

The *depth* of a propositional formula θ is the maximum number of nested conjunctions or disjunctions in θ. Note that the definitions of Γ_t and Δ_t are purely syntactical; every formula in Γ_t or Δ_t is equivalent to a formula in Γ_0. But the translation from a formula in Γ_t to an equivalent formula in Γ_0 usually increases the depth of a formula. For all $t, d \geq 0$ we let $\Gamma_{t,d}$ denote the class of all formulas in Γ_t whose small subformulas have depth at most d (equivalently, we may say that the whole formula has depth at most $d + t$). We define $\Delta_{t,d}$ analogously.

Let CNF denote the class of all propositional formulas in conjunctive normal form, that is, conjunctions of disjunctions of literals; if we ignore arbitrarily nested negations, then CNF is just $\Gamma_{2,0}$. A formula is in *d conjunctive normal form* if it is a conjunction of disjunctions of at most d literals; the class of all such formulas is denoted by d-CNF.

The *weight* of a truth value assignment to the variables of a propositional formula is the number of variables set to TRUE by the assignment. For any class Θ of propositional formulas, the *weighted satisfiability problem for* Θ, denoted by $\mathrm{WSAT}(\Theta)$, is the problem of deciding whether a formula in Θ has a satisfying assignment of weight k, parameterized by k. We are now ready to define the W-hierarchy as follows.

DEFINITION 2.2. *For* $t \geq 1$, $\mathrm{W}[t]$ *is the class of all parameterized problems that can be reduced to* $\mathrm{WSAT}(\Gamma_{t,d})$ *for some* $d \geq 0$ *by a parameterized many-one reduction.*

We omit the definition of parameterized many-one reductions here and refer the reader to [13] for this definition and further background on parameterized complexity theory.

2.2. Relational structures. A *vocabulary* is a finite set of relation symbols. Associated with every relation symbol is a natural number, its *arity*. The arity of a

vocabulary is the maximum of the arities of the relation symbols it contains. In the following, τ always denotes a vocabulary.

A *τ-structure* $\mathcal{A}$ consists of a nonempty set A, called the *universe* of $\mathcal{A}$, and a relation $R^{\mathcal{A}} \subseteq A^r$ for each r-ary relation symbol $R \in \tau$. For example, we view a *directed graph* as a structure $\mathcal{G} = (G, E^{\mathcal{G}})$ whose vocabulary consists of one binary relation symbol E. $\mathcal{G} = (G, E^{\mathcal{G}})$ is an *(undirected) graph* if $E^{\mathcal{G}}$ is symmetric. For graphs, we often write $\{a, b\} \in E^{\mathcal{G}}$ instead of $(a, b) \in E^{\mathcal{G}}$. In this paper, we consider only structures whose universe is finite. We distinguish between the size of the universe A of a τ-structure $\mathcal{A}$, which we denote by $|A|$, and the *size* of $\mathcal{A}$, which is defined to be

$$\|\mathcal{A}\| = |\tau| + |A| + \sum_{R \in \tau} |R^{\mathcal{A}}| \cdot \text{arity}(R).$$

An *expansion* of a τ-structure $\mathcal{A}$ to a vocabulary $\tau' \supseteq \tau$ is a τ'-structure $\mathcal{A}'$ with $A' = A$ and $R^{\mathcal{A}'} = R^{\mathcal{A}}$ for all $R \in \tau$.

A *substructure* of $\mathcal{A}$ is a structure $\mathcal{B}$ with $B \subseteq A$ and $R^{\mathcal{B}} \subseteq R^{\mathcal{A}}$ for all $R \in \tau$.[1] A *homomorphism* from a τ-structure $\mathcal{A}$ to a τ-structure $\mathcal{B}$ is a mapping $h : A \to B$, where for all $R \in \tau$, say, of arity r, and all tuples $(a_1, \ldots, a_r) \in R^{\mathcal{A}}$ we have $\big(h(a_1), \ldots, h(a_r)\big) \in R^{\mathcal{B}}$. An *embedding* is a homomorphism that is one-to-one.

The *homomorphism problem* asks whether there is a homomorphism from a given structure $\mathcal{A}$ to a given structure $\mathcal{B}$. We parameterize this problem by the size of $\mathcal{A}$ and denote the resulting *parameterized homomorphism problem* by p-Hom. We will also consider the *parameterized embedding problem*, denoted by p-Emb, and the *parameterized substructure problem* (Does structure $\mathcal{B}$ have a substructure isomorphic to $\mathcal{A}$?), denoted by p-Sub. Of course when considered as decision problems, p-Emb and p-Sub are equivalent, but as counting problems they are slightly different. All three decision problems are complete for the class W[1] under parameterized many-one reductions [13].

2.3. Logic and descriptive complexity. Let us remark that the following notions are not needed for understanding our results on the hardness of counting cycles and paths or their proofs.

The formulas of *first-order logic* are built up from *atomic formulas* using the usual Boolean connectives and existential and universal quantification over the elements of the universe of a structure. Remember that an *atomic formula*, or *atom*, is a formula of the form $x = y$ or $Rx_1 \ldots x_r$, where R is an r-ary relation symbol and $x, y, x_1, \ldots, x_r$ are variables. A *literal* is either an atom or a negated atom. The *vocabulary* of a formula φ is the set of all relation symbols occurring in φ. A *free variable* of a formula φ is a variable that is not bound by any existential or universal quantifier of φ.

If $\mathcal{A}$ is a τ-structure, $a_1, \ldots, a_n$ are elements of the universe A of $\mathcal{A}$, and $\varphi(x_1, \ldots, x_n)$ is a formula whose vocabulary is a subset of τ and whose free variables are $x_1, \ldots, x_n$, then we write $\mathcal{A} \models \varphi(a_1, \ldots, a_n)$ to denote that $\mathcal{A}$ satisfies φ if the variables $x_1, \ldots, x_n$ are interpreted by $a_1, \ldots, a_n$, respectively. We let

$$\varphi(\mathcal{A}) := \big\{(a_1, \ldots, a_n) \in A^n \;\big|\; \mathcal{A} \models \varphi(a_1, \ldots, a_n)\big\}.$$

[1] Note that in logic, substructures are usually required to satisfy the stronger condition $R^{\mathcal{B}} = R^{\mathcal{A}} \cap A^r$, where r is the arity of R. Our notion of substructure is the direct generalization of the standard graph theoretic notion of subgraph. Since we are dealing mainly with graphs, this seems appropriate. A similar remark applies to our notion of embedding.

To get a uniform notation, we let A^0 be a one-point space and identify $\emptyset$ with FALSE and A^0 with TRUE. Then for a sentence φ (i.e., a formula without free variables), we have $\mathcal{A} \models \varphi \iff \varphi(\mathcal{A}) = \text{TRUE}$. Furthermore, if the vocabulary of the formula φ is not contained in the vocabulary of $\mathcal{A}$, then we let $\varphi(\mathcal{A}) = \emptyset$.

For every class Φ of formulas, we let $\Phi[\tau]$ be the class of all $\varphi \in \Phi$ whose vocabulary is contained in τ. We let both Σ_0 and Π_0 be the class of all quantifier-free first-order formulas (although we usually use Π_0 to denote this class). For $t \geq 1$, we let Σ_t be the class of all first-order formulas of the form $\exists x_1 \dots \exists x_k\ \psi$, where $k \in \mathbb{N}$ and $\psi \in \Pi_{t-1}$. Analogously, we let Π_t be the class of all first-order formulas of the form $\forall x_1 \dots \forall x_k\ \psi$, where $k \in \mathbb{N}$ and $\psi \in \Sigma_{t-1}$.

We have to define two additional hierarchies $(\Sigma_{t,u})_{t\geq 1}$ and $(\Pi_{t,u})_{t\geq 1}$ for every fixed $u \geq 1$. Again we let $\Sigma_{0,u} = \Pi_{0,u} = \Pi_0$. We let $\Pi_{1,u}$ be the class of all first-order formulas of the form $\forall x_1 \dots \forall x_k\ \psi$, where $k \leq u$ and $\psi \in \Pi_0$. For $t \geq 2$, we let $\Pi_{t,u}$ be the class of all first-order formulas of the form $\forall x_1 \dots \forall x_{k_1}\ \exists y_1 \dots \exists y_{k_2}\ \psi$, where $k_1, k_2 \leq u$ and $\psi \in \Pi_{t-2,u}$. For $t \geq 1$, we let $\Sigma_{t,u}$ be the class of all first-order formulas of the form $\exists x_1 \dots \exists x_k\ \psi$, where $k \in \mathbb{N}$ and $\psi \in \Pi_{t-1,u}$. Note the asymmetry in the definitions of $\Pi_{t,u}$ and $\Sigma_{t,u}$—the length of the first quantifier block in a $\Sigma_{t,u}$-formula is not restricted.

Definability of parameterized problems I: *Model-checking problems.* We can use logic to define certain generic families of parameterized problems. For a class Φ of formulas, the *model-checking problem* for Φ is the problem of deciding whether for a given structure $\mathcal{A}$ and a given formula $\varphi \in \Phi$ we have $\varphi(\mathcal{A}) \neq \emptyset$. We parameterize this problem by the length of the formula φ and obtain the *parameterized model-checking problem* p-MC(Φ).

Many parameterized problems can be naturally translated into model-checking problems. For example, the parameterized clique problem is essentially the same as the parameterized model-checking problem for the class

$$\Phi_{\text{CLIQUE}} = \left\{ \bigwedge_{1\leq i<j\leq k} (Ex_ix_j \wedge x_i \neq x_j) \;\middle|\; k \geq 1 \right\}.$$

Model-checking problems provide another basis for the W-hierarchy: For every $t \geq 1$, W[t] is the class of all problems that are reducible to p-MC($\Sigma_{t,1}[\tau]$) for some vocabulary τ by a parameterized many-one reduction [14, 15]. Observe, furthermore, that for all $t \geq 1$ the problems p-MC($\Sigma_{t,1}$) and p-MC($\Pi_{t-1,1}$) are easily reducible to each other, because for every formula

$$\varphi(x_1, \dots, x_k) = \exists y_1 \dots \exists y_l\ \psi(x_1, \dots, x_k, y_1, \dots, y_l)$$

and every structure $\mathcal{A}$ we have $\varphi(\mathcal{A}) \neq \emptyset$ if and only if $\psi(\mathcal{A}) \neq \emptyset$. This explains why the hierarchies $(\Sigma_{t,u})_{t\geq 1}$ and $(\Pi_{t,u})_{t\geq 1}$ are defined asymmetrically.

Definability of parameterized problems II: *Fagin-definability.* There is a second way of defining parameterized problems that has been dubbed *Fagin-definability* in [15]. Let φ be a sentence of vocabulary $\tau \cup \{X\}$, where X is a relation symbol not contained in τ. We view X as a *relation variable*; to illustrate this we usually write $\varphi(X)$ instead of just φ. Let r be the arity of X. For a τ-structure $\mathcal{A}$ we let

$$\varphi(\mathcal{A}) = \big\{ R \subseteq A^r \;\big|\; (\mathcal{A}, R) \models \varphi \big\},$$

where $(\mathcal{A}, R)$ denotes the $\tau \cup \{X\}$-expansion of $\mathcal{A}$ with $X^{(\mathcal{A},R)} = R$. For example, let X be unary and

$$\varphi_{\text{VC}}(X) = \forall y \forall z \big(Eyz \to (Xy \vee Xz) \big).$$

```
1  Initialize S ⊆ Pow(G) by S := {∅}
2  for all {a, b} ∈ E^G do
3      for all S ∈ S do
4          if S ∩ {a, b} = ∅ then
5              S := S \ {S}
6              if |S| < k then S := S ∪ {S ∪ {a}, S ∪ {b}}.
7  output S.
```

ALGORITHM 1

Then for a graph $\mathcal{G}$, $\varphi_{\mathrm{VC}}(\mathcal{G})$ is the set of all vertex covers of $\mathcal{G}$.

With each formula $\varphi(X)$ we associate a parameterized problem p-FD($\varphi(X)$) which asks whether for a given structure $\mathcal{A}$, the set $\varphi(\mathcal{A})$ contains a relation with k elements (where k is the parameter). We call p-FD($\varphi(X)$) the problem *Fagin-defined* by $\varphi(X)$.

For example, p-FD($\varphi_{\mathrm{VC}}(X)$) is precisely the parameterized vertex cover problem.

3. Tractable parameterized counting problems. A *parameterized counting problem* is simply a function $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$, for some alphabet Σ. Arguably, this definition includes problems that we would not intuitively call counting problems, but there is no harm in including them.

DEFINITION 3.1. *A parameterized counting problem $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ is* fixed-parameter tractable*, or $F \in \mathrm{FPT}$, if there is an algorithm computing $F(x, k)$ in time $f(k) \cdot |x|^c$ for some computable function $f : \mathbb{N} \to \mathbb{N}$ and some constant $c \in \mathbb{N}$.*

The standard example of a fixed-parameter tractable decision problem is the parameterized version of the vertex cover problem. As a first example, we observe that the corresponding counting problem is also fixed-parameter tractable, as shown in the following example.

Example 3.2. The parameterized vertex cover counting problem

p-#VC	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the number of vertex covers of $\mathcal{G}$ of size k.

is fixed-parameter tractable.

Proof. Essentially, Algorithm 1 is the standard procedure showing that the parameterized vertex cover problem is fixed-parameter tractable. It yields, given a graph $\mathcal{G} = (G, E^{\mathcal{G}})$ as input and $k \in \mathbb{N}$ as parameter, a set $\mathcal{S}$ of subsets of cardinality $\leq k$ of G in time $O(2^k \cdot \|\mathcal{G}\|)$ such that for the set $\mathrm{VC}_k(\mathcal{G})$ of vertex covers of $\mathcal{G}$ of cardinality k we have

$$\mathrm{VC}_k(\mathcal{G}) = \{X \subseteq G \mid |X| = k \text{ and } S \subseteq X \text{ for some } S \in \mathcal{S}\}.$$

Now, we can compute $|\mathrm{VC}_k(\mathcal{G})|$ by applying the inclusion-exclusion principle to the sets

$$\{X \subseteq G \mid |X| = k \text{ and } S \subseteq X\}$$

for $S \in \mathcal{S}$. ∎

We could now go through a list of known fixed-parameter tractable problems and check if the corresponding counting problems are also fixed-parameter tractable. Fortunately, this boring task can largely be avoided, because there are a few general principles underlying most fixed-parameter tractability results. They are formulated in the terminology of descriptive complexity theory.

(1) *Problems definable in monadic second-order logic are fixed-parameter tractable when parameterized by tree-width of the structure* (Courcelle [9]). This accounts for the fixed-parameter tractability of NP-complete problems such as 3-Colorability or Hamiltonicity when parameterized by the tree-width of the input graph.

(2) *Parameterized problems that can be described as model-checking problems for first-order logic are fixed-parameter tractable on classes of structures of bounded local tree-width and classes of graphs with excluded minors* (Frick and Grohe [17], Flum and Grohe [15]). This implies that parameterized versions of problems such as dominating set, independent set, or subgraph isomorphism are fixed-parameter tractable on planar graphs or on graphs of bounded degree.

(3) *Parameterized problems that are Fagin-definable by a first-order formula $\varphi(X)$, where X does not occur in the scope of a negation symbol or existential quantifier, are fixed-parameter tractable* (Cai and Chen [7], Flum and Grohe [15]). This accounts for the fixed-parameter tractability of the standard parameterization of minimization problems in the classes MIN $\mathrm{F}^{+}\Pi_1$ [21], for example, minimum vertex cover.

(4) *Parameterized problems that can be described as parameterized model-checking problems for Σ_1-formulas of bounded tree-width are fixed-parameter tractable* (Flum and Grohe [15]). This implies, and is actually equivalent to, the results that the parameterized homomorphism problem and the parameterized embedding problem for relational structures of bounded tree-width are fixed-parameter tractable.

Let us consider the counting versions of these general "meta-theorems." It has already been proved by Arnborg, Lagergren, and Seese [4] that the counting version of (1) holds. Variants and extensions of this result have been proved by Courcelle, Makowsky, and Rotics [10] and Makowsky [22]. Frick proved that (most of) (2) also extends to counting problems [16]. We shall see below that (3) also extends to counting problems. (4) is more problematic. While the counting version of the parameterized homomorphism problem for structures of bounded tree-width is fixed-parameter tractable (actually in polynomial time), the general equivalence between homomorphism, embedding, and model-checking for Σ_1 breaks down for counting problems. This is the point where counting shows some genuinely new aspects, and in some sense, most of this paper is devoted to this phenomenon.

Let us turn to (3), the Fagin-definable problems. For a formula $\varphi(X)$, we let p-#FD($\varphi(X)$) denote the natural counting version of the problem p-FD($\varphi(X)$) Fagin-defined by $\varphi(X)$. Recalling the formula $\varphi_{\mathrm{VC}}(X)$ that Fagin-defines the parameterized vertex cover problem, we see that the following proposition generalizes Example 3.2.

Proposition 3.3. *Let $\varphi(X)$ be a first-order formula in which X does not occur in the scope of an existential quantifier or negation symbol. Then p-#FD($\varphi(X)$) $\in$ FPT.*

Proof. Let X be of arity r. As for the vertex cover problem one obtains an FPT-algorithm (cf. [15]) that, given a structure $\mathcal{A}$ and $k \in \mathbb{N}$, yields a set $\mathcal{S}$ of subsets of cardinality $\leq k$ of A^r such that

$\{X \subseteq A^r \mid |X| = k \text{ and } \mathcal{A} \models \varphi(X)\} = \{X \subseteq A^r \mid |X| = k \text{ and } S \subseteq X \text{ for some } S \in \mathcal{S}\}.$

Again, an application of the inclusion-exclusion principle allows us to compute the cardinality of the set on the right-hand side. □

This implies that the counting versions of the standard parameterizations of all minimization problems in the class MIN $\mathrm{F}^+\Pi_1$ are fixed-parameter tractable.

As we have mentioned, the situation with item (4) in the list above is more complicated. The core problem for which counting remains tractable is the homomorphism problem for graphs of bounded tree-width. Again, the algorithm showing tractability can best be illustrated by an example.

Example 3.4. The number of homomorphisms from a given colored tree $\mathcal{T}$ to a given colored graph $\mathcal{G}$ can be computed in polynomial time.

This can be done by a simple dynamic programming algorithm. Starting from the leaves, for every vertex t of the tree we compute a table that stores, for all vertices v of the graph, the number $H(t, v)$ of homomorphisms h from $\mathcal{T}_t$, the induced colored subtree rooted at t, to $\mathcal{G}$ with $h(t) = v$. Then the total number of homomorphisms from $\mathcal{T}$ to $\mathcal{G}$ is $\sum_{v \in G} H(r, v)$, where r is the root of $\mathcal{T}$.

If t is a leaf, then $H(t, v) = 1$ if t and v have the same color and $H(t, v) = 0$ otherwise. If t has children $t_1, \dots, t_l$, then if t and v have the same color we have

$$H(t, v) = \prod_{i=1}^{l} \sum_{\substack{w \in G \\ w \text{ adjacent to } v}} H(t_i, w).$$

If t and v have distinct colors, we have $H(t, v) = 0$.

The previous example can easily be generalized to structures of bounded tree-width. We just state the result, omit a definition of tree-width, and omit the proof, which is a straightforward generalization of the example.

PROPOSITION 3.5. *Let $w \geq 1$. Then the following restriction of the homomorphism problem is in polynomial time:*

Input:	Structure $\mathcal{A}$ of tree-width at most w, structure $\mathcal{B}$.
Problem:	Count the homomorphisms from $\mathcal{A}$ to $\mathcal{B}$.

For a class Φ of formulas, we let $p\text{-}\#(\Phi)$ denote the counting version of the model-checking problem $p\text{-MC}(\Phi)$ ("given $\mathcal{A}$ and $\varphi \in \Phi$, compute $|\varphi(\mathcal{A})|$, parameterized by $|\varphi|$").

With every first-order formula φ we associate a graph $\mathcal{G}_\varphi$ as follows: The vertices of $\mathcal{G}_\varphi$ are the variables of φ, and there is an edge between two vertices if they occur together in an atomic subformula of φ. The *tree-width* of a formula φ is the tree-width of $\mathcal{G}_\varphi$. For a class Φ of formulas and $w \geq 1$, we let $\Phi[\mathrm{tw}\, w]$ denote the class of all formulas in Φ of tree-width at most w. Recall that Π_0 denotes the class of all quantifier-free formulas.

PROPOSITION 3.6. *For every $w \geq 1$ we have $p\text{-}\#(\Pi_0[\mathrm{tw}\, w]) \in \mathrm{FPT}$.*

Proof. We can effectively transform every $\varphi(\bar{x}) \in \Pi_0[\mathrm{tw}\, w]$ into an equivalent $\psi(\bar{x}) \in \Pi_0[\mathrm{tw}\, w]$ in disjunctive normal form, $\psi(\bar{x}) = \psi_1(\bar{x}) \vee \cdots \vee \psi_r(\bar{x})$, where each $\psi_i(\bar{x})$ is a conjunction of literals and where $\psi_i(\bar{x}) \wedge \psi_j(\bar{x})$ is unsatisfiable for all i, j with $i \neq j$. Note that this transformation does not change the tree-width of the formula because the set of atomic subformulas remains unchanged. Then for every structure $\mathcal{A}$ we have

$$|\psi(\mathcal{A})| = |\psi_1(\mathcal{A})| + \cdots + |\psi_r(\mathcal{A})|.$$

Thus we can restrict our attention to formulas in $\Pi_0[\operatorname{tw} w]$ that are conjunctions of literals. Since every literal of a formula whose underlying graph has tree-width at most w contains at most $w+1$ variables, by standard techniques (cf. the proof of Theorem 4.4) the counting problem for such formulas can be reduced to the counting version of the homomorphism problem for structures of tree-width at most w, which is in polynomial time by Proposition 3.5. □

Remark 3.7. Note that although its core is a reduction to Proposition 3.5, the proof of the previous proposition does *not* yield a polynomial time algorithm. The reason is that the transformation of a formula to an equivalent formula in disjunctive normal form is not polynomial.

Indeed, it is easy to see that the unparameterized counting problem for quantifier-free formulas of tree-width 0 is #P-complete.

Clearly, if p-MC(Φ) is fixed-parameter tractable, then so is p-MC(Φ^*), where Φ^* is the closure of Φ under existential quantification. Thus in particular, for $w \geq 1$ the problem p-MC($\Sigma_1[\operatorname{tw} w]$) is fixed-parameter tractable. The situation is different for the counting problems: The formula

$$\varphi(x_1, \ldots, x_k) := \exists y \bigwedge_{i=1}^{k} (\neg E y x_i \wedge \neg y = x_i)$$

is a Σ_1-formula of tree-width 1. For all graphs $\mathcal{G}$, the set $\varphi(\mathcal{G})$ is the set of all tuples $(a_1, \ldots, a_k)$ of vertices of $\mathcal{G}$ such that $\{a_1, \ldots, a_k\}$ is *not* a dominating set. Thus $\mathcal{G}$ has a dominating set of size at most k if and only if $|\varphi(\mathcal{G})| < n^k$, where n is the number of vertices of $\mathcal{G}$. Since the parameterized dominating set problem is complete for the class W[2], this implies the following:

PROPOSITION 3.8. *If* W[2] $\neq$ FPT*, then* p-#($\Sigma_1[\operatorname{tw} w]$) $\notin$ FPT.

4. Classes of intractable problems.

Example 4.1. Valiant's [29] fundamental theorem states that counting the number of perfect matchings of a bipartite graph is #P-complete (whereas deciding whether a perfect matching exists is in P). We consider a *trivial parameterization* of the matching problem, which is obtained by adding a "dummy" parameter as follows:

Input:	Bipartite Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ has a perfect matching.

Clearly, this problem is in polynomial time and thus fixed-parameter tractable. On the other hand, its counting version ("Count the perfect matchings of $\mathcal{G}$") cannot be fixed-parameter tractable unless P = #P. The reason for this is that the problem is already #P-complete for the fixed parameter value $k = 1$, but if it were fixed-parameter tractable it would be in polynomial time for any fixed parameter value.

Of course this example is quite artificial. We are more interested in the question of whether natural parameterized counting problems are fixed-parameter tractable. As examples of such natural problems we mention p-#CLIQUE ("Count cliques of size k in a graph, where k is the parameter"), p-#DOMINATING SET ("Count dominating sets of size k"), p-#CYCLE ("Count cycles of size k"), or, as a more natural parameterization of the matching problem, p-#MATCHING ("Count the matchings of size k in a bipartite graph"). An argument such as the one in Example 4.1 cannot be used to show that any of these problems is not fixed-parameter tractable, because for any fixed parameter value k the problems are in polynomial time.

Recall that the decision problems p-Clique and p-Dominating Set are complete for the classes W[1] and W[2], respectively, so the counting problems cannot be fixed-parameter tractable unless W[1] = FPT (W[2] = FPT, respectively). We will define classes #W[t] of counting problems and show for a few central W[1]-complete and W[2]-complete problems that their counting versions are #W[1]-complete (#W[2]-complete, respectively). More interestingly, in the next section we shall prove that p-#Cycle and a number of similar problems whose decision versions are fixed-parameter tractable are complete for #W[1].

Definition 4.2. *Let $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ and $G : \Pi^* \times \mathbb{N} \to \mathbb{N}$ be parameterized counting problems.*

(1) *A* parameterized parsimonious reduction *from F to G is an algorithm that computes for every instance (x, k) of F an instance (y, ℓ) of G in time $f(k) \cdot |x|^c$ such that $\ell \leq g(k)$ and*

$$F(x, k) = G(y, \ell)$$

(for computable functions $f, g : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$).
We write $F \leq^{\mathrm{fp}}_{\mathrm{pars}} G$ to denote that there is a parameterized parsimonious reduction from F to G.

(2) *A* parameterized T-reduction *from F to G is an algorithm with an oracle for G that solves any instance (x, k) of F in time $f(k) \cdot |x|^c$ in such a way that for all oracle queries the instances (y, ℓ) satisfy $\ell \leq g(k)$ (for computable functions $f, g : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$).*
We write $F \leq^{\mathrm{fp}}_{\mathrm{T}} G$ to denote that there is a parameterized T-reduction from F to G.

Obviously, if $F \leq^{\mathrm{fp}}_{\mathrm{pars}} G$, then $F \leq^{\mathrm{fp}}_{\mathrm{T}} G$. An easy computation shows that if $G \in \mathrm{FPT}$ and $F \leq^{\mathrm{fp}}_{\mathrm{T}} G$, then $F \in \mathrm{FPT}$.

For a class Θ of propositional formulas, we let #WSat(Θ) be the counting version of the weighted satisfiability problem for Θ ("Count the weight k satisfying assignments for a formula $\theta \in \Theta$"). We define the counting analogue of the W-hierarchy in the following straightforward way:

Definition 4.3. *For $t \geq 1$, #W[t] is the class of all parameterized counting problems that are fixed-parameter parsimonious reducible to #WSat($\Gamma_{t,d}$), for some $d \geq 0$.*

The notation #W[t] may be slightly misleading when compared with the notation #P of classical complexity theory (which is not #NP), but since there is no obvious #FPT, we think that it is appropriate. Note that we write FPT to denote both the class of fixed-parameter tractable decision problems and the class of fixed-parameter tractable counting problems; the intended meaning will always be clear from the context.

4.1. #W[1]-complete problems.

Theorem 4.4. *The following problems are complete for #W[1] under parameterized parsimonious reductions:*

(1) *#WSat(2-CNF),*
(2) *p-#Clique, p-#Sub, p-#Hom, p-#Emb,*
(3) *p-#($\Pi_0[\tau]$) for every vocabulary τ that is not monadic,*
(4) *p-#Halt ("Count the k-step accepting computation paths of a nondeterministic Turing machine").*

Proof. Basically, the proof of these results amounts to checking that the many-one reductions proving the W[1]-completeness of the corresponding decision problems are

parsimonious (or can be made parsimonious by simple modifications). Some of these reductions are quite simple, and we can sketch them here. For those that are more complicated, we just give appropriate references.

A *conjunctive query* is a first-order formula of the form $\exists x_1 \dots \exists x_k(\alpha_1 \wedge \dots \wedge \alpha_\ell)$, where $\alpha_1, \dots, \alpha_\ell$ are atoms. In particular, a *quantifier-free* conjunctive query is just a conjunction of atoms. We denote the class of all conjunctive queries by CQ and the class of all quantifier-free conjunctive queries by Π_0-CQ. If Φ is a class of formulas, then Φ[binary] is the class of all formulas $\varphi \in \Phi$ whose vocabulary is at most binary.

We will first establish the following chain of reductions:

$$p\text{-}\#(\Pi_0\text{-CQ}) \leq^{\text{fp}}_{\text{pars}} p\text{-}\#\textsc{Hom} \leq^{\text{fp}}_{\text{pars}} p\text{-}\#\textsc{Emb} \leq^{\text{fp}}_{\text{pars}} p\text{-}\#(\Pi_0\text{-CQ}). \tag{4.1}$$

p-$\#(\Pi_0$-CQ$) \leq^{\text{fp}}_{\text{pars}} p$-$\#$Hom: With every formula $\varphi(x_1, \dots, x_k) \in \Pi_0$-CQ of vocabulary τ we associate a $\tau \cup \{\text{EQ}\}$-structure $\mathcal{A}_\varphi$, where EQ is a binary relation symbol not contained in τ. The universe of $\mathcal{A}_\varphi$ is $\{x_1, \dots, x_k\}$, the set of variables of φ. For $R \in \tau$, say, r-ary, $R^{\mathcal{A}_\varphi}$ is the set of all tuples $(x_{i_1}, \dots, x_{i_r})$ such that $Rx_{i_1} \dots x_{i_r}$ is an atom of φ. Moreover, $\text{EQ}^{\mathcal{A}_\varphi}$ is the set of all pairs (x_{i_1}, x_{i_2}) such that $x_{i_1} = x_{i_2}$ is an atom of φ. Note that $\|\mathcal{A}_\varphi\| \in O(|\varphi|)$.

For a τ-structure $\mathcal{B}$ we let $\mathcal{B}_{\text{EQ}}$ be the $\tau \cup \{\text{EQ}\}$-expansion of $\mathcal{B}$ in which EQ is interpreted by the equality relation on B. Then it is easy to see that for all $(b_1, \dots, b_k) \in B^k$ we have $(b_1, \dots, b_k) \in \varphi(\mathcal{B})$ if and only if the mapping $x_i \mapsto b_i$, for $1 \leq i \leq k$, is a homomorphism from $\mathcal{A}_\varphi$ to $\mathcal{B}$. This yields a parsimonious reduction from p-$\#(\Pi_0$-CQ$)$ to p-$\#$Hom.

p-$\#$Hom $\leq^{\text{fp}}_{\text{pars}} p$-$\#$Emb: Suppose we have structures $\mathcal{A}$ and $\mathcal{B}$ and want to count the homomorphisms from $\mathcal{A}$ to $\mathcal{B}$. Let τ be the vocabulary of $\mathcal{A}$ and $\mathcal{B}$ and $\tau^* = \tau \cup \{P_a \mid a \in A\}$, where for every $a \in A$, P_a is a new unary relation symbol that is not contained in τ. Let $\mathcal{A}^*$ be the τ^*-expansion of $\mathcal{A}$ with $P_a = \{a\}$ for $a \in A$. We can view $\mathcal{A}^*$ as the expansion of $\mathcal{A}$ where each element gets its individual color. We let $\mathcal{B}^*$ be the following τ^*-structure: The universe of $\mathcal{B}^*$ is $A \times B$. For r-ary $R \in \tau$ and $(a_1, b_1), \dots, (a_r, b_r) \in A \times B$ we let $\big((a_1, b_1), \dots, (a_r, b_r)\big) \in R^{\mathcal{B}^*}$ if and only if $(b_1, \dots, b_r) \in R^{\mathcal{B}}$. For $a \in A$ let $P_a^{\mathcal{B}^*} = \{a\} \times B$. For a homomorphism $h : \mathcal{A} \to \mathcal{B}$ we let $h^* : A^* \to B^*$ be the mapping defined by $h^*(a) = (a, h(a))$. It is easy to see that the mapping $h \mapsto h^*$ is a bijection between the homomorphisms from $\mathcal{A}$ to $\mathcal{B}$ and the embeddings from $\mathcal{A}^*$ to $\mathcal{B}^*$.

p-$\#$Emb $\leq^{\text{fp}}_{\text{pars}} p$-$\#(\Pi_0$-CQ$)$: For every τ-structure $\mathcal{A}$ we define a formula $\varphi_{\mathcal{A}} \in \Pi_0$-CQ of vocabulary $\tau \cup \{\text{NEQ}\}$, where NEQ is a new binary relation symbol. Suppose that $\mathcal{A} = \{a_1, \dots, a_k\}$. The formula $\varphi_{\mathcal{A}}$ has variables $x_1, \dots, x_k$. For every r-ary $R \in \tau$ and every tuple $(a_{i_1}, \dots, a_{i_r}) \in R^{\mathcal{A}}$, $\varphi_{\mathcal{A}}$ contains the atom $Rx_{i_1} \dots x_{i_r}$. In addition, $\varphi_{\mathcal{A}}$ contains the atoms $\text{NEQ}x_i x_j$ for $1 \leq i < j \leq k$.

For a τ-structure $\mathcal{B}$ we let $\mathcal{B}_{\text{NEQ}}$ be the $\tau \cup \{\text{NEQ}\}$-expansion of $\mathcal{B}$ in which NEQ is interpreted by the inequality relation on B. Then it is easy to see that for all mappings $h : A \to B$ we have that h is an embedding of $\mathcal{A}$ into $\mathcal{B}$ if and only if $\big(h(a_1), \dots, h(a_k)\big) \in \varphi_{\mathcal{A}}(\mathcal{B})$.

Next, we establish the following chain of reductions (for every $d \geq 1$):

$$\begin{aligned} \#\text{WSat}(\Gamma_{1,d}) &\leq^{\text{fp}}_{\text{pars}} p\text{-}\#(\Pi_0\text{-CQ}) \leq^{\text{fp}}_{\text{pars}} p\text{-}\#(\Pi_0\text{-CQ[binary]}) \\ &\leq^{\text{fp}}_{\text{pars}} p\text{-}\#\textsc{Clique} \leq^{\text{fp}}_{\text{pars}} \#\text{WSat(2-CNF)} \leq^{\text{fp}}_{\text{pars}} \#\text{WSat}(\Gamma_{1,1}). \end{aligned} \tag{4.2}$$

Together with (4.1), this proves the #W[1]-completeness of all problems listed in (1) and (2) except p-$\#$Sub.

$\#\mathrm{WSAT}(\Gamma_{1,d}) \leq^{\mathrm{fp}}_{\mathrm{pars}} p\text{-}\#(\Pi_0\text{-CQ})$: The proof of Lemma 21 in [19] showing that $\mathrm{WSAT}(\Gamma_{1,d})$ is fixed-parameter many-one reducible to p-MC(CQ) yields a parsimonious reduction from $\#\mathrm{WSAT}(\Gamma_{1,d})$ to $p\text{-}\#(\Pi_0\text{-CQ})$.

$p\text{-}\#(\Pi_0\text{-CQ}) \leq^{\mathrm{fp}}_{\mathrm{pars}} p\text{-}\#(\Pi_0\text{-CQ[binary]})$: The proof of Lemma 17 in [19] showing that p-MC(CQ) is fixed-parameter many-one reducible to p-MC(CQ[binary]) yields the claimed parsimonious reduction. Here and in later proofs we use (variants of) the following observation: Let $\varphi(\bar{x})$ and $\psi(\bar{x},\bar{y})$ be formulas and $\mathcal{A}$ a structure. If for all tuples $\bar{a} \in A$ we have

$$\mathcal{A} \models \varphi(\bar{a}) \iff \mathcal{A} \models \exists \bar{y} \psi(\bar{a},\bar{y}),$$

and for all tuples $\bar{a} \in A$ there exists at most one tuple $\bar{b} \in A$ such that $\mathcal{A} \models \psi(\bar{a},\bar{b})$, then $|\varphi(\mathcal{A})| = |\psi(\mathcal{A})|$.

$p\text{-}\#(\Pi_0\text{-CQ[binary]}) \leq^{\mathrm{fp}}_{\mathrm{pars}} p\text{-}\#\mathrm{CLIQUE}$: The reduction in Proposition 22 of [19] is parsimonious.

$p\text{-}\#\mathrm{CLIQUE} \leq^{\mathrm{fp}}_{\mathrm{pars}} \#\mathrm{WSAT}(\text{2-CNF})$: Let $\mathcal{G}$ be a graph. For every $a \in G$ let X_a be a propositional variable. Set

$$\alpha_{\mathcal{G}} = \bigwedge_{a,b \in G,\, a \neq b,\, (a,b) \notin E^{\mathcal{G}}} (\neg X_a \vee \neg X_b) \;\wedge\; \bigwedge_{a \in G} (X_a \vee \neg X_a).$$

Then $\alpha_{\mathcal{G}}$ is (equivalent to) a formula in 2-CNF. The second part of the formula ensures that every variable X_a with $a \in G$ occurs in $\alpha_{\mathcal{G}}$. The number of cliques of size k is just the number of assignments of weight k satisfying $\alpha_{\mathcal{G}}$.

$\#\mathrm{WSAT}(\text{2-CNF}) \leq^{\mathrm{fp}}_{\mathrm{pars}} \#\mathrm{WSAT}(\Gamma_{1,1})$: 2-CNF is a subset of $\Gamma_{1,1}$, so the reduction is trivial.

This completes the proof of (4.2). We next show (3). Let τ be a vocabulary that is not monadic. We leave it to the reader to show that $p\text{-}\#\mathrm{CLIQUE} \leq^{\mathrm{fp}}_{\mathrm{pars}} p\text{-}\#(\Pi_0[\tau])$.

$p\text{-}\#(\Pi_0[\tau]) \leq^{\mathrm{fp}}_{\mathrm{pars}} p\text{-}\#(\Pi_0\text{-CQ})$: Let $\mathcal{A}$ be a τ-structure and $\varphi \in \Pi_0[\tau]$. We can assume that $\varphi = \varphi_1 \vee \cdots \vee \varphi_m$, where each φ_i is a conjunction of literals and $\varphi_i \wedge \varphi_j$ is unsatisfiable for all $i \neq j$. Let $\tau' := \tau \cup \{\bar{R} \mid R \in \tau\} \cup \{\mathrm{EQ}, \mathrm{NEQ}\}$, where for all $R \in \tau$ the symbol $\bar{R}$ is a new relation symbol of the same arity as R and $\mathrm{EQ}, \mathrm{NEQ}$ are new binary relation symbols. Let $\mathcal{A}'$ be the τ'-expansion of $\mathcal{A}$ in which $\bar{R}$ is interpreted as the complement of $R^{\mathcal{A}}$ and EQ and NEQ are interpreted as equality and inequality, respectively. Since the vocabulary is fixed, $\mathcal{A}'$ can be computed from $\mathcal{A}$ in polynomial time. If n is the size of the universe of $\mathcal{A}$, then computing the relations $\mathrm{EQ}^{\mathcal{A}'}$ and $\mathrm{NEQ}^{\mathcal{A}'}$ requires quadratic time, and for an r-ary $R \in \tau$, computing $\bar{R}^{\mathcal{A}'} = A^r \setminus R^{\mathcal{A}}$ requires time $O(n^r)$.

Let φ' be the formula obtained by replacing positive literals of the form $x = y$ by $\mathrm{EQ}xy$ and by replacing negative literals by positive ones in the obvious way using the new relation symbols $\bar{R}$ and NEQ. Then $\varphi' = \varphi'_1 \vee \cdots \vee \varphi'_m$, where each φ'_i is a conjunction of atoms (i.e., positive literals). Note that $\varphi(\mathcal{A}) = \varphi'(\mathcal{A}')$ and

(4.3) for $\bar{a} \in \mathcal{A}'$ there is at most one i with $\mathcal{A}' \models \varphi'_i(\bar{a})$.

Finally we want to get rid of the disjunctions in φ'. For this purpose we introduce a structure $\mathcal{A}''$ essentially consisting of m copies of $\mathcal{A}'$, the ith one taking care of φ'_i. More precisely, let $\tau'' := \{\hat{R} \mid R \in \tau'\} \cup \{<, T\}$, where $\mathrm{arity}(\hat{R}) = \mathrm{arity}(R) + 1$ and where $<$ and T are binary. Define the τ''-structure $\mathcal{A}''$ by

$$
\begin{aligned}
A'' &:= \{1,\dots,m\} \cup (\{1,\dots,m\} \times A),\\
<^{\mathcal{A}''} &:= \text{the natural ordering on } \{1,\dots,m\},\\
T^{\mathcal{A}''} &:= \{((i,a),(i,b)) \mid 1 \le i \le m,\ a,b \in A\},\\
\hat{R}^{\mathcal{A}''} &:= \{(i,(i,a_1),\dots,(i,a_{\mathrm{arity}(R)})) \mid 1 \le i \le m,\ R^{\mathcal{A}'}a_1\dots a_{\mathrm{arity}(R)}\}\\
&\quad \cup \{(i,(j,a_1),\dots,(j,a_{\mathrm{arity}(R)})) \mid 1 \le i,j \le m,\ i \ne j,\ a_1,\dots,a_{\mathrm{arity}(R)} \in A\}.
\end{aligned}
$$

Moreover, set

$$
\varphi''(x_1,\dots,x_k,y_1,\dots,y_m) := y_1 < \dots < y_m \wedge \bigwedge_{1\le \ell \le \ell' \le k} Tx_\ell x_{\ell'} \wedge \bigwedge_{i=1}^{m} \varphi_i'^{\frac{\hat{R}y_i\bar{z}}{R\bar{z}}},
$$

where $\varphi_i'^{\frac{\hat{R}y_i\bar{z}}{R\bar{z}}}$ is obtained from φ_i' by replacing, for all $R \in \tau'$, atomic subformulas of the form $R\bar{z}$ by $\hat{R}y_i\bar{z}$. Clearly, $\varphi(\bar{x},\bar{y}) \in \Pi_0$-CQ. By (4.3), we have $|\varphi(\mathcal{A})| = |\varphi''(\mathcal{A}'')|$.

Next, we prove the #W[1]-completeness of p-#Sub. We observe that the number of substructures of a structure $\mathcal{B}$ that are isomorphic to a structure $\mathcal{A}$ equals the number of embeddings of $\mathcal{A}$ into $\mathcal{B}$ divided by the number of automorphisms of $\mathcal{A}$. Unfortunately, this does not immediately yield a parsimonious reduction from p-#Sub to p-#Emb or vice versa. However, p-#Clique is a restriction of p-#Sub; thus we have p-#Clique $\le^{\mathrm{fp}}_{\mathrm{pars}}$ p-#Sub.

To prove that p-#Sub is in #W[1], we reduce p-#Sub to p-#Emb. Let $\mathcal{A},\mathcal{B}$ be τ-structures and let $<$ be a binary relation symbol not contained in τ. Let us call a $\tau \cup \{<\}$-structure $\mathcal{C}$, in which $<^{\mathcal{C}}$ is a linear order of the universe, an *ordered* $\tau \cup \{<\}$-*structure*. Let $\mathcal{A}_1,\dots,\mathcal{A}_m$ be a list of expansions of $\mathcal{A}$ to ordered $\tau \cup \{<\}$-structures such that

(i) for $1 \le i < j \le m$, the structures $\mathcal{A}_i$ and $\mathcal{A}_j$ are not isomorphic,
(ii) every expansion $\mathcal{A}'$ of $\mathcal{A}$ to an ordered $\tau \cup \{<\}$-structure is isomorphic to an $\mathcal{A}_i$ for some i, $1 \le i \le m$.

Thus $\mathcal{A}_1,\dots,\mathcal{A}_m$ is a list of all ordered expansions of $\mathcal{A}$, where each isomorphism type is listed only once.

Let $\mathcal{B}_<$ be an arbitrary expansion of $\mathcal{B}$ to an ordered $\tau \cup \{<\}$-structure. Then

$$
\Big|\{\mathcal{A}' \subseteq \mathcal{B} \mid \mathcal{A}' \cong \mathcal{A}\}\Big| = \sum_{i=1}^{m} \Big|\{\mathcal{A}' \subseteq \mathcal{B}_< \mid \mathcal{A}' \cong \mathcal{A}_i\}\Big|.
$$

Moreover, for each i the number of substructures of $\mathcal{B}_<$ isomorphic to $\mathcal{A}_i$ is equal to the number of embeddings of $\mathcal{A}_i$ into $\mathcal{B}_<$.

Let $\prec$ be another binary relation symbol not contained in $\tau \cup \{<\}$ and let $\tau^* = \tau \cup \{<,\prec\}$. Let $\mathcal{A}^*$ be the τ^*-structure obtained by taking the disjoint union of $\mathcal{A}_1,\dots,\mathcal{A}_m$ and defining $\prec^{\mathcal{A}}$ such that for all $a_i \in A_i$, $a_j \in A_j$ we have $a_i \prec^{\mathcal{A}^*} a_j$ if and only if $i < j$. For $1 \le i \le m$, let $\mathcal{B}_i^*$ be the τ^*-structure obtained by replacing the copy of $\mathcal{A}_i$ in $\mathcal{A}^*$ by a copy of $\mathcal{B}_<$. Then the number of embeddings of $\mathcal{A}^*$ into $\mathcal{B}_i^*$ is equal to the number of embeddings of $\mathcal{A}_i$ into $\mathcal{B}_<$. Finally let $\mathcal{B}^*$ be the disjoint union of $\mathcal{B}_1^*,\dots,\mathcal{B}_m^*$. Then the number of embeddings of $\mathcal{A}^*$ into $\mathcal{B}^*$ is equal to the sum of the numbers of embeddings of $\mathcal{A}^*$ into $\mathcal{B}_i^*$ for $1 \le i \le m$. Putting everything together, the number of substructures of $\mathcal{B}$ isomorphic to $\mathcal{A}$ is equal to the number of embeddings of $\mathcal{A}^*$ into $\mathcal{B}^*$.

It remains to prove #W[1]-completeness of p-#Halt. The proof of Theorem 8.3 in [15] implicitly contains parsimonious reductions from p-#(Π_0-CQ[binary]) to p-#Halt and from p-#Halt to p-#(Π_0-CQ). □

The decision versions of all problems mentioned in Theorem 4.4 are W[1]-complete under parameterized many-one reductions. The following theorem is interesting because it is not known whether the decision problem p-MC(Π_0) is contained in the closure of W[1] under parameterized T-reductions.

THEOREM 4.5. *p-#(Π_0) is contained in the closure of #W[1] under parameterized T-reductions.*

Proof. We shall prove that p-#(Π_0) $\leq_{\mathrm{T}}^{\mathrm{fp}}$ p-#(Π_0-CQ).

Note that the reduction from p-#($\Pi_0[\tau]$) to p-#(Π_0-CQ) we gave in the proof of Theorem 4.4 does not yield a parameterized parsimonious reduction from p-#(Π_0) to p-#(Π_0-CQ), because if the vocabulary is not fixed in advance the structure $\mathcal{A}'$ can get much larger than $\mathcal{A}$.

At least, the same argument as given in the proof of Theorem 4.4 shows that we can restrict our attention to conjunctions of literals (instead of arbitrary quantifier-free formulas). Consider a formula

$$\varphi = \alpha_1 \wedge \cdots \wedge \alpha_\ell \wedge \neg\beta_1 \wedge \cdots \wedge \neg\beta_m,$$

where $\alpha_1, \ldots, \alpha_\ell, \beta_1, \ldots, \beta_m$ are atoms. The crucial observation is that for any structure $\mathcal{A}$ we have

$$\begin{aligned} |\varphi(\mathcal{A})| = {} & \big|(\alpha_1 \wedge \cdots \wedge \alpha_\ell \wedge \neg\beta_1 \wedge \cdots \wedge \neg\beta_{m-1})(\mathcal{A})\big| \\ & - \big|(\alpha_1 \wedge \cdots \wedge \alpha_\ell \wedge \neg\beta_1 \wedge \cdots \wedge \neg\beta_{m-1} \wedge \beta_m)(\mathcal{A})\big|. \end{aligned}$$

Note that the two formulas on the left-hand side of the equality have fewer negated atoms than φ. We can now recursively reduce the number of negated atoms in these two formulas using the same trick until we end up with a family of quantifier-free conjunctive queries. This gives us a parameterized Turing reduction from p-#(Π_0) to p-#(Π_0-CQ). □

4.2. A machine characterization of #W[1]. As it is also the case for many other parameterized complexity classes, the definition of the classes #W[t] is a bit unsatisfactory because all the classes are defined only as the closure of a certain problem under a certain type of reduction. In particular, one may ask why we chose parsimonious reductions and not, say, Turing reductions. Indeed, McCartin [23] defined her version of the classes #W[t] using a different form of reductions, and that makes the theory seem a bit arbitrary. Compare this with the situation for the class #P, which has a natural machine characterization: A classical counting problem $F : \Sigma^* \to \mathbb{N}$ is in #P if and only if there is a polynomial time nondeterministic Turing machine N such that for every instance x of the problem, $F(x)$ is the number of accepting paths of N on input x.

Recently, a machine characterization of the class W[1] was given [8]. In this subsection, we adapt this characterization to give a characterization of #W[1] along the lines of the above mentioned characterization of #P.

The machine model we use, which has been introduced in [8], is based on the standard random access machines (RAMs) described in [25]. The arithmetic operations are addition, subtraction, and division by 2 (rounded off), and we use a uniform cost measure. The model is nonstandard when it comes to nondeterminism. A *nondeterministic RAM* is a RAM with an additional instruction "GUESS i j" whose semantics is "guess a natural number less than or equal to the number stored in register i and store it in register j." Acceptance of an input by a nondeterministic RAM program is defined as usually for nondeterministic machines. Steps of a computation of a

nondeterministic RAM that execute a GUESS instruction are called *nondeterministic steps*.

Following [8], we call a nondeterministic RAM program $\mathbb{P}$ a W-*program*, if there is a computable function f and a polynomial p such that for every input (x,k) with $|x| = n$ the program $\mathbb{P}$ on every run

(1) performs at most $f(k) \cdot p(n)$ steps;
(2) performs at most $f(k)$ nondeterministic steps;
(3) uses at most the first $f(k) \cdot p(n)$ registers;
(4) contains numbers $\leq f(k) \cdot p(n)$ in all registers at any time.

We call a W-program $\mathbb{P}$ a W[1]-*program* if there is a computable function h such that for every input (x,k), for every run of $\mathbb{P}$

(5) all nondeterministic steps are among the last $h(k)$ steps.

THEOREM 4.6 (Chen, Flum, and Grohe [8]). *Let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized decision problem. Then $Q \in$ W[1] if and only if there is a* W[1]*-program deciding Q.*

The main result of this section is a counting version of Theorem 4.6.

THEOREM 4.7. *Let $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ be a parameterized counting problem. Then $F \in \#$W[1] if and only if there is a* W[1]*-program $\mathbb{P}$ such that, for all $(x,k) \in \Sigma^* \times \mathbb{N}$, $F(x,k)$ is the number of accepting paths of $\mathbb{P}$ on input (x,k).*

Proof. First assume that $F \in \#$W[1]. Then, by Theorem 4.4, there is a parsimonious reduction from F to p-#HALT. Hence, there are computable functions f, g, a polynomial p, and an algorithm assigning to every instance (x,k) of F, in time $\leq f(k) \cdot p(n)$, a nondeterministic Turing machine $M = M_{x,k}$ and a natural number $k' = k'(x,k) \leq g(k)$ such that $F(x,k)$ is the number of accepting paths of M of length k'.

We can assume that the states and the symbols of the alphabet of M are natural numbers $\leq f(k) \cdot p(n)$. We define a W-program $\mathbb{P}$ that on input $(x,k) \in \Sigma^* \times \mathbb{N}$ proceeds as follows:

1. It computes M and k';
2. It guesses a sequence of k' configurations of M;
3. It verifies that the sequence of guessed configurations forms an accepting computation of M.

We can do this, in particular line 1, with a W-program using our parameterized parsimonious reduction from F to p-#HALT. Moreover, the number of steps needed by lines 2 and 3 is bounded by $h(k)$ for a suitable computable function h. Finally, the number of accepting paths of $\mathbb{P}$ is exactly the number of accepting paths of M.

Assume now that we have a W[1]-program $\mathbb{P}$ such that for all $(x,k) \in \Sigma^* \times \mathbb{N}$, $F(x,k)$ is the number of accepting paths of $\mathbb{P}$ on input (x,k). Let f, p, h witness that $\mathbb{P}$ is a W[1]-program. For every instance $(x,k) \in \Sigma^* \times \mathbb{N}$ of F we shall define a nondeterministic Turing machine $M = M_{x,k}$ and an integer k' such that $F(x,k)$ is the number of accepting paths of M of length at most k'. Of course we have to do this in such a way that the mapping $(x,k) \mapsto (M,k')$ is a parameterized reduction.

So let $(x,k) \in \Sigma^* \times \mathbb{N}$ and $n = |x|$. The alphabet of $M = M_{x,k}$ contains $0, 1, \ldots, f(k) \cdot p(n)$. Thus alphabet symbols can be used to represent register content and register addresses of all runs of $\mathbb{P}$ on input (x,k). In addition, the alphabet contains a few control symbols. The transition function of M will be defined in such a way that M simulates the computation of $\mathbb{P}$ on input (x,k) *from the first nondeterministic step onwards*. The content of all the registers before the first nondeterministic step is hardwired into M. The changes of the register contents during the at most $h(k)$ nondeterministic steps are written on the worktape, so eventually the worktape contains pairs $(i_1, a_1), \ldots, (i_\ell, a_\ell)$ in any order, where (i_j, a_j) indicates

that the current content of register i_j is a_j, and $\ell \leq h(k)$. For more details on the definition of M we refer the reader to [8]. □

4.3. #W[1] and counting satisfying assignments of a 3-CNF-formula. The following theorem gives further evidence that #W[1] ≠ FPT, because it seems unlikely that counting the satisfying assignments of a 3-CNF-formula with n variables is possible in time $2^{o(n)}$. A decision version of this theorem has been proved by Abrahamson, Downey, and Fellows [1].

THEOREM 4.8. *If* #W[1] = FPT*, then there is an algorithm counting the satisfying assignments of a* 3-CNF*-formula with n variables in time $2^{o(n)}$.*

Proof. Suppose that #W[1] = FPT. Then #WSAT(3-CNF) is in FPT. Thus there is an algorithm solving #WSAT(3-CNF) in time $f(k) \cdot n^c$ for some computable function $f : \mathbb{N} \to \mathbb{N}$ and constant c. Then there exists a function $g : \mathbb{N} \to \mathbb{N}$ such that

(i) $f(g(n)) \leq 2^{o(n)}$,
(ii) $\lim_{n\to\infty} g(n) = \infty$,
(iii) $g(n)$ can be computed in time $2^{o(n)}$.

Let $\gamma = \bigwedge_{i=1}^{m} \delta_i$, where each clause δ_i is a disjunction of at most three literals, be a formula in 3-CNF, and let $\mathcal{X} = \{X_1, \ldots, X_n\}$ be the set of variables of γ. We assume that no clause appears twice; thus we have $m \leq (2n)^3$. We want to compute the number of satisfying assignments of γ in time $2^{o(n)}$.

Let $k = g(n)$. Note that (ii) implies $n/k \leq o(n)$; we will use this repeatedly in the following argument. For $1 \leq j \leq k$, let

$$\mathcal{X}_j = \Big\{ X_i \;\Big|\; (j-1) \cdot \frac{n}{k} < i \leq j \cdot \frac{n}{k} \Big\}.$$

For every $S \subseteq \mathcal{X}_j$, let Y_j^S be a new variable. Let $\mathcal{Y}_j$ be the set of all Y_j^S and $\mathcal{Y} = \bigcup_{j=1}^{k} \mathcal{Y}_j$. Then

$$|\mathcal{Y}| \leq k \cdot 2^{\lceil n/k \rceil} \leq 2^{o(n)}.$$

Call a truth value assignment to the variables in $\mathcal{Y}$ *good* if for $1 \leq j \leq k$ exactly one variable in $\mathcal{Y}_j$ is set to TRUE. There is a bijection I between the truth value assignments to the variables in $\mathcal{X}$ and the good truth value assignments to the variables in $\mathcal{Y}$ defined by

$$I(A)(Y_j^S) = \text{TRUE} \iff \forall X \in \mathcal{X}_j : \big(A(X) = \text{TRUE} \iff X \in S\big),$$

for all $A : \mathcal{X} \to \{\text{TRUE}, \text{FALSE}\}$, $1 \leq j \leq k$, and $S \subseteq \mathcal{X}_j$. Let

$$\beta = \bigwedge_{\substack{1 \leq j \leq k \\ S,T \subseteq \mathcal{X}_j,\, S \neq T}} (\neg Y_j^S \vee \neg Y_j^T)$$

and note that $|\beta| \leq k \cdot \big(2^{\lceil n/k \rceil}\big)^2 \leq 2^{o(n)}$. Observe that the weight k assignments to the variables in $\mathcal{Y}$ satisfying β are precisely the good assignments. Thus there is a bijection between the weight k satisfying assignments for β and the assignments to the variables in $\mathcal{X}$.

For $1 \leq j \leq k$ and every variable $X \in \mathcal{X}_j$, let

$$\alpha_X = \bigwedge_{S \subseteq \mathcal{X}_j,\, X \notin S} \neg Y_j^S,$$

$$\alpha_{\neg X} = \bigwedge_{S \subseteq \mathcal{X}_j,\, X \in S} \neg Y_j^S$$

and observe that for every assignment $A : \mathcal{X} \to \{\text{TRUE}, \text{FALSE}\}$ we have

$$\begin{aligned} A(X) = \text{TRUE} &\iff I(A) \text{ satisfies } \alpha_X \\ &\iff I(A) \text{ does not satisfy } \alpha_{\neg X}. \end{aligned}$$

Let γ' be the formula obtained from γ by replacing each literal X by the formula α_X and each literal $\neg X$ by $\alpha_{\neg X}$. Then for every assignment $A : \mathcal{X} \to \{\text{TRUE}, \text{FALSE}\}$ we have

$$A \text{ satisfies } \gamma \iff I(A) \text{ satisfies } \gamma'.$$

By applying de Morgan's rule to each clause δ_j of γ (or rather to the disjunction of conjunctions δ_j has become in γ') we can turn γ' into an equivalent conjunction of at most

$$m \cdot \left(2^{\lceil n/k \rceil}\right)^3$$

disjunctions of at most three literals each. Let γ'' be this 3-CNF-formula and let $\gamma^* = \beta \wedge \gamma''$. Then I is a bijection between the satisfying assignments of γ and the weight k satisfying assignments of γ^*.

By our initial assumption, we can compute the number of weight k satisfying assignments of γ^* in time $f(k) \cdot (n^*)^c$, where $n^* = |\mathcal{Y}| \leq 2^{o(n)}$ is the number of variables of γ^*. Since $f(k) = f(g(n)) \leq 2^{o(n)}$, this shows that we can compute the number of satisfying assignments of γ in time $2^{o(n)}$. □

4.4. #W[2]-complete problems.

THEOREM 4.9. *The following problems are complete for* #W[2] *under parameterized parsimonious reductions:*

(1) #WSAT(CNF),

(2) *p*-#DOMINATING SET,

(3) *p*-#($\Pi_{1,1}[\tau]$) *for every vocabulary* τ *that is not monadic.*

The equivalence between (1) and (3) in Theorem 4.9 can be lifted to the other classes of the #W-hierarchy, but we deal only with #W[2] here.

Proof of Theorem 4.9. Let $\Pi_{1,1}[s]$ denote the class of all formulas in $\Pi_{1,1}$ whose vocabulary is at most s-ary. We will establish the following chain of reductions for every $d \geq 0$ and $s \geq 2$:

$$\#\text{WSAT}(\Gamma_{2,d}) \leq^{\text{fp}}_{\text{pars}} p\text{-}\#(\Pi_{1,1}[s]) \leq^{\text{fp}}_{\text{pars}} p\text{-}\#\text{DOMINATING SET} \leq^{\text{fp}}_{\text{pars}} \#\text{WSAT(CNF)}.$$

Recalling that CNF $\subseteq \Gamma_{2,0}$ and observing that p-#DOMINATING SET $\leq^{\text{fp}}_{\text{pars}}$ p-#($\Pi_{1,1}[\tau]$) for every vocabulary τ that is not monadic, we see that this proves the theorem.

#WSAT($\Gamma_{2,d}$) $\leq^{\text{fp}}_{\text{pars}}$ p-#($\Pi_{1,1}[2]$): By standard means one can show that there is a d' depending only on d such that every formula in $\Gamma_{2,d}$ is equivalent to a formula of the form

$$\alpha = \bigwedge_{i \in I} \delta_i,$$

where for some $d_\alpha \leq d'$ every δ_i is a disjunction of conjunctions of exactly d_α literals,

$$\delta_i = \bigvee_{j \in J^i} \beta_{ij}$$

with

$$\beta_{ij} = \lambda_{ij1} \wedge \cdots \wedge \lambda_{ijd_\alpha}. \tag{4.4}$$

So let such an α, say, with variables $X_1, \dots, X_n$, and a $k \in \mathbb{N}$ be given. If we have an assignment of weight k setting $X_{i_1}, \dots, X_{i_k}$ with $i_1 < \cdots < i_k$ TRUE and satisfying β_{ij} as in (4.4), then the positive literals in β_{ij} must be among $X_{i_1}, \dots, X_{i_k}$. Thus for every negative literal $\neg X_r$ in β_{ij} we must have $r < i_1$ or $i_k < r$ or $i_s < r < i_{s+1}$ for some s. We use this fact in our reduction appropriately.

For $m \in \mathbb{N}$ set $[m] := \{1, \dots, m\}$ and

$$[m]_2 := \{(i,j) \mid 0 \le i < j \le m+1\}.$$

For a set M and $m \in \mathbb{N}$ denote by $\mathrm{Pow}_{\le m}(M) := \{Y \subseteq M \mid |Y| \le m\}$ the set of subsets of M of cardinality $\le m$.

We let $\tau = \{<, \prec, E, \mathrm{FIRST}, \mathrm{LAST}, F, \mathrm{DISJ}, \mathrm{SAT}\}$ with binary $<, \prec, E, F, \mathrm{SAT}$ and unary FIRST, LAST, DISJ. Let $\mathcal{A}_\alpha$ be the following τ-structure: The universe is

$$A_\alpha := [n] \cup [n]_2 \cup \mathrm{Pow}_{\le d_\alpha}([n] \cup [n]_2) \cup \{\delta_i \mid i \in I\}.$$

Recall that I is the index set of the conjunction in the formula α. The selection of $i \in [n]$ means that the variable X_i gets the value TRUE and the selection of $(i,j) \in [n]_2$ means that all variables X_ℓ with $i < \ell < j$ get the truth value FALSE.

The relations of $\mathcal{A}_\alpha$ are specified by

$$\begin{aligned}
<^{\mathcal{A}_\alpha} &:= \text{the natural ordering on } [n];\\
\prec^{\mathcal{A}_\alpha} &:= \text{a total (“lexicographic”) ordering on } \mathrm{Pow}_{\le d_\alpha}([n] \cup [n]_2);\\
E^{\mathcal{A}_\alpha} &:= \{(j,(i,j)) \mid 0 \le i < j \le n+1\} \cup \{(i,(i,j)) \mid 0 \le i < j \le n+1\};\\
\mathrm{FIRST}^{\mathcal{A}_\alpha} &:= \{(0,j) \mid 0 \le j \le n+1\};\\
\mathrm{LAST}^{\mathcal{A}_\alpha} &:= \{(i,n+1) \mid 0 \le i \le n+1\};\\
F^{\mathcal{A}_\alpha} &:= \{(i,M) \mid i \in [n],\ M \in \mathrm{Pow}_{\le d_\alpha}([n] \cup [n]_2),\ i \in M\}\\
&\quad \cup \{((i,j),M) \mid (i,j) \in [n]_2,\ M \in \mathrm{Pow}_{\le d_\alpha}([n] \cup [n]_2),\ (i,j) \in M\};\\
\mathrm{DISJ}^{\mathcal{A}_\alpha} &:= \{\delta_i \mid i \in I\};\\
\mathrm{SAT}^{\mathcal{A}_\alpha} &:= \{(M,\delta_i) \mid M \in \mathrm{Pow}_{\le d_\alpha}([n] \cup [n]_2),\ i \in I,\\
&\qquad \text{there is a } j \in J^i \text{ such that for } s = 1, \dots, n,\\
&\qquad\quad \text{if } X_s \text{ is a literal of } \beta_{ij}, \text{ then } s \in M \text{ and}\\
&\qquad\quad \text{if } \neg X_s \text{ is a literal of } \beta_{ij}, \text{ then there is } (\ell,m) \in M \text{ with } \ell < s < m\}.
\end{aligned}$$

Let $r := |\mathrm{Pow}_{\le d_\alpha}([2 \cdot k+1])|$. Note that $\|\mathcal{A}_\alpha\| \le \|\mathcal{A}\|^c$, where $c = c(d)$, and $r \le g(d,k)$ for some computable function g.

The number of satisfying assignments of α of weight k is $|\varphi_{\alpha,k}(\mathcal{A}_\alpha)|$, where $\varphi_{\alpha,k}(x_1, \dots, x_k, z_1, \dots, z_{k+1}, u_1, \dots, u_r)$ is the $\Pi_{1,1}$-formula

$$\begin{aligned}
\varphi_{\alpha,k} = \forall y \Big(& x_1 < \cdots < x_k \wedge \textstyle\bigwedge_{i=1}^k (Ex_iz_i \wedge Ex_iz_{i+1}) \wedge \mathrm{FIRST}\, z_1 \wedge \mathrm{LAST}\, z_{k+1}\\
& \wedge u_1 \prec \cdots \prec u_r \wedge \textstyle\bigwedge_{i=1}^r \Big(Fyu_i \to \big(\bigvee_{j=1}^k y = x_j \vee \bigvee_{j=1}^{k+1} y = z_j \big) \Big)\\
& \wedge \big(\mathrm{DISJ}\, y \to \textstyle\bigvee_{j=1}^r \mathrm{SAT}\, u_j y \big) \Big).
\end{aligned}$$

$p\text{-}\#(\Pi_{1,1}[s]) \leq^{\text{fp}}_{\text{pars}} p\text{-}\#\text{DOMINATING SET}$: For notational simplicity, we assume $s = 2$. Let τ be a vocabulary that contains only unary and binary relation symbols. Assume we are given a τ-structure $\mathcal{A}$ with universe A and a $\Pi_{1,1}[\tau]$-formula

$$\varphi(x_1, \ldots, x_\ell) = \forall y \psi(x_1, \ldots, x_\ell, y).$$

Let $\mathcal{G} = (G, E^{\mathcal{G}})$ be the graph defined as follows: The vertex set is

$$\begin{aligned} G := {} & (\{1, \ldots, \ell\} \times A) \,\dot\cup\, A^\ell \,\dot\cup\, (A \times \{0\}) \\ & \dot\cup\, \{b_i^j \mid 1 \leq i \leq \ell,\ 1 \leq j \leq \ell + 2\} \\ & \dot\cup\, \{b^j \mid 1 \leq j \leq \ell + 2\} \end{aligned}$$

($\dot\cup$ denotes disjoint union), where b_i^j and b^j are new elements. The edge relation $E^{\mathcal{G}}$ is defined in such a way that

(i) every $(a_1, \ldots, a_\ell) \in A^\ell$ is connected to all elements of $\{i\} \times (A \setminus \{a_i\})$ for $1 \leq i \leq \ell$;
(ii) for $\bar{a} \in A^\ell$ and $(b, 0) \in A \times \{0\}$: $\{\bar{a}, (b, 0)\} \in E^{\mathcal{G}} \iff \mathcal{A} \models \psi(\bar{a}, b)$;
(iii) b_i^j is connected to (i, a) for $1 \leq i \leq \ell$, $1 \leq j \leq \ell + 2$, $a \in A$;
(iv) b^j is connected to all $\bar{a} \in A^\ell$ for $1 \leq j \leq \ell + 2$.

We claim that

- Every dominating set of $\mathcal{G}$ of cardinality $\ell + 1$ contains exactly one element of each $\{i\} \times A$, and if we label these elements, say, (i, a_i) for $1 \leq i \leq \ell$, then the $(\ell + 1)$st element in the dominating set is $(a_1, \ldots, a_\ell) \in A^\ell$.
- For all $a_1, \ldots, a_\ell \in A$,

$$\big\{(i, a_i) \;\big|\; 1 \leq i \leq \ell\big\} \cup \{\bar{a}\} \text{ is a dominating set of } \mathcal{G} \iff \mathcal{A} \models \varphi(\bar{a}).$$

To see this, suppose that D is a dominating set of $\mathcal{G}$ of size $\ell + 1$. Then D must contain at least one vertex of $\{i\} \times A$ for $1 \leq i \leq \ell$ and one vertex of A^ℓ, because this is the only way the vertices b_i^j and b^j, for $1 \leq j \leq \ell + 2$, can be dominated with $\ell + 1$ vertices. Suppose that D contains the vertices $(1, a_1), \ldots, (\ell, a_\ell)$. Let d be the remaining element of D. If $d = (d_1, \ldots, d_\ell) \neq (a_1, \ldots, a_\ell)$, say, $d_1 \neq a_1$, then $(1, d_1)$ is not dominated by $a_1, \ldots, a_\ell, d$. Therefore, d must be $(a_1, \ldots, a_\ell)$. However, d must also dominate $A \times \{0\}$, and this is possible only if $\mathcal{A} \models \psi(a_1, \ldots, a_\ell, b)$ for all $b \in A$.

Thus $|\varphi(\mathcal{A})|$ is the number of dominating sets of $\mathcal{G}$ of cardinality $\ell + 1$. But note that $\mathcal{G}$ is too big for a parameterized reduction, since G contains the set A^ℓ, where the exponent depends on the parameter φ. So we need a more refined reduction. We can assume that $\psi(\bar{x}, y) = \psi_1 \wedge \cdots \wedge \psi_m$ where each ψ_i is a disjunction of literals. Each literal contains at most two variables. Therefore, we do not need A^ℓ but a copy A_{ij} of A^2 for $1 \leq i < j \leq \ell$. We replace A^ℓ above by all these copies and $A \times \{0\}$ by $A \times \{1, \ldots, m\}$. We replace (i) by

(i′) every (a_i, a_j) in the copy A_{ij} of A^2 is connected to all elements of $\{i\} \times (A \setminus \{a_i\})$ and all elements of $\{j\} \times (A \setminus \{a_j\})$.

Furthermore, we replace (ii) by

(ii′) for $1 \leq i < j \leq \ell$, (a_i, a_j) in the copy A_{ij} of A^2, and for $(b, k) \in A \times \{1, \ldots, m\}$,

$$\{(a_i, a_j), (b, k)\} \in E^{\mathcal{G}} \iff \begin{array}{l}\text{there is a literal } \lambda(x_i, x_j, y) \text{ in } \psi_k \text{ whose (at most} \\ \text{two) free variables are among } x_i, x_j, y \text{ such that} \\ \mathcal{A} \models \lambda(a_i, a_j, b).\end{array}$$

Moreover, instead of the b_i^j and the b^j we add for every $i = 1, \dots, \ell$ and for every copy of A^2 a set of $\ell + \binom{\ell}{2} + 1$ new elements that ensure that every dominating set of cardinality $\ell + \binom{\ell}{2}$ contains exactly one element of every $\{i\} \times A$ and of every copy of A^2.

Then dominating sets of cardinality $\ell + \binom{\ell}{2}$ and tuples in $\mathcal{A}$ satisfying φ are related in a one-to-one fashion.

p-#DOMINATING SET $\leq^{\text{fp}}_{\text{pars}}$ #WSAT(CNF): Let $\mathcal{G} = (G, E^{\mathcal{G}})$ be a graph. For $a \in G$ let X_a be a propositional variable. Let $\alpha_{\mathcal{G}}$ be the propositional formula

$$\alpha_{\mathcal{G}} := \bigwedge_{a \in G} \left(X_a \vee \bigvee_{(a,b) \in E} X_b \right).$$

Then, $\alpha_{\mathcal{G}}$ is (equivalent to) a formula in $\Gamma_{2,0}$. Clearly the number of satisfying assignments of $\alpha_{\mathcal{G}}$ of weight k equals the number of dominating sets of $\mathcal{G}$ of size k. □

Remark 4.10. As opposed to the proof of Theorem 4.4, the reductions given in the proof of Theorem 4.9 are not just variants of the standard reductions showing the W[2]-completeness of the respective problems under many-one reductions. As a matter of fact, our proof yields a new proof of the complicated result that p-DOMINATING SET is W[2]-complete under parameterized many-one reductions.

5. Counting cycles and paths.

THEOREM 5.1. *The following problems are* #W[1]*-complete under parameterized Turing reductions:*

(1) p-#CYCLE *and* p-#DIRCYCLE
("Count the cycles of length k in a (directed) graph").
(2) p-#PATH *and* p-#DIRPATH
("Count the paths of length k in a (directed) graph").

To be precise, let us define a *path of length* k in a directed graph $(G, E^{\mathcal{G}})$ to be a substructure of G isomorphic to $\big(\{1, \dots, k\}, \{(i, i+1) \mid 1 \leq i < k\}\big)$. A *cycle of length* k is a substructure isomorphic to $\big(\{1, \dots, k\}, \{(i, i+1) \mid 1 \leq i < k\} \cup \{(k, 1)\}\big)$. Paths and cycles in undirected graphs are defined similarly.

Thus all problems in Theorem 5.1 are restrictions of the substructure problem p-#SUB and thus in #W[1] by Theorem 4.4. The decision versions of the problems are fixed-parameter tractable. This is an immediate consequence of Plehn and Voigt's [26] theorem that the parameterized embedding problem restricted to graphs of bounded tree-width is fixed-parameter tractable and the fact that paths have tree-width 1 and cycles have tree-width 2.

LEMMA 5.2.

$$p\text{-\#DIRCYCLE} \leq^{\text{fp}}_{\text{pars}} p\text{-\#CYCLE} \leq^{\text{fp}}_{\text{T}} p\text{-\#PATH} \leq^{\text{fp}}_{\text{T}} p\text{-\#DIRPATH}.$$

Proof. p-#DIRCYCLE $\leq^{\text{fp}}_{\text{pars}}$ p-#CYCLE: For a directed graph $\mathcal{G}$, let $\mathcal{G}^u_{p,q}$ be the undirected graph obtained from $\mathcal{G}$ by the following two steps:

(1) Replace each vertex a of $\mathcal{G}$ by an undirected path of length p such that the (directed) edges with head a in $\mathcal{G}$ get the first vertex of this path as their new head and the edges with tail a in $\mathcal{G}$ get the last vertex of this path as their new tail.
(2) Replace each directed edge in this graph (corresponding to an edge of $\mathcal{G}$) by an undirected path of length q.

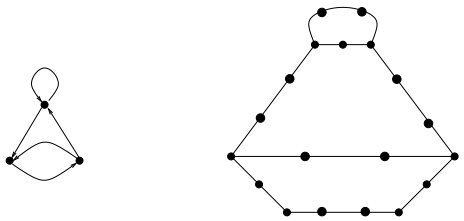


FIG. 5.1. *A directed graph $\mathcal{G}$ and the corresponding $\mathcal{G}^u_{2,3}$.*

Figure 5.1 gives an example.

Observe that each cycle in $\mathcal{G}^u_{p,q}$ has length $\ell \cdot p + m \cdot q$ for some integers $\ell, m \geq 0$ with $\ell \leq m$. Further observe that each directed cycle of length k in $\mathcal{G}$ lifts to a cycle of length $k(p+q)$ in $\mathcal{G}^u_{p,q}$. Given k, we want to choose p and q in such a way that each cycle of length $k(p+q)$ in $\mathcal{G}^u_{p,q}$ is the lifting of a directed cycle of length k in $\mathcal{G}$. To achieve this, we have to choose p and q in such a way that

$$k(p+q) \neq \ell \cdot p + m \cdot q \tag{5.1}$$

for all $\ell, m \geq 0$ with $\ell < m$. If we choose $p \leq q$, then (5.1) holds for $m > 2k$. So we have to fulfill (5.1) for $0 \leq \ell < m \leq 2k$. Hence, we have to avoid $\binom{2k+1}{2}$ linear equalities. Clearly we can find natural numbers $p \leq q$ satisfying none of these equalities.

For such p and q, the number of directed cycles of length k in $\mathcal{G}$ equals the number of undirected cycles of length $k(p+q)$ in $\mathcal{G}^u_{p,q}$.

p-#CYCLE $\leq^{\text{fp}}_{\text{T}}$ p-#PATH: Let $\mathcal{G}$ be an undirected graph and $k \geq 1$. Without loss of generality we can assume that $k \geq 3$ because counting loops in a graph is easy.

For each $e = \{v, w\} \in E^{\mathcal{G}}$ and all $\ell, m \geq 0$, we let $\mathcal{G}_e(\ell, m)$ be the graph obtained from $\mathcal{G}$ by adding vertices as $v_1, \ldots, v_\ell, w_1, \ldots, w_m$ and adding edges between v_i and w for $1 \leq i \leq \ell$ and between w_j and v for $1 \leq j \leq m$.

We observe that the number x_e of paths of length $k+1$ from v_1 to w_1 in $\mathcal{G}_e(\ell, m)$ is exactly the number of cycles of length k in $\mathcal{G}$ containing the edge e. We now show how to compute x_e from the numbers of paths of length $k+1$ in the graphs $\mathcal{G}_e(\ell, m)$ for $0 \leq \ell, m \leq 1$. This yields a parameterized Turing reduction from p-#CYCLE to p-#PATH.

We observe that the v_i and w_j can only be endpoints of paths in $\mathcal{G}_e(\ell, m)$, and that each path can have at most one endpoint among $v_1, \ldots, v_\ell$ and at most one endpoint among $w_1, \ldots, w_m$ (because each path ending in v_i must go through w and each path ending in w_j must go through v).

We let

- $x = x_e$ be the number of paths of length $(k+1)$ from v_1 to w_1 in $\mathcal{G}_e(1,1)$,
- y be the number of paths of length $(k+1)$ in $\mathcal{G}_e(1,1)$ that contain v_1 but not w_1,
- z be the number of paths of length $(k+1)$ in $\mathcal{G}_e(1,1)$ that contain w_1 but not v_1,
- w be the number of paths of length $(k+1)$ in $\mathcal{G}_e(1,1)$ that contain neither v_1 nor w_1.

Let $p_{\ell m}$ be the number of paths of length $(k+1)$ in $\mathcal{G}_e(\ell, m)$. Then we have

$$p_{\ell m} = w + \ell \cdot m \cdot x + \ell \cdot y + m \cdot z.$$

For $0 \leq \ell, m \leq 1$ we obtain a system of four linear equations in the variables w, x, y, z whose matrix is nonsingular. Thus it has a unique solution which, in particular, gives us the desired value x.

p-#Path $\leq_{\mathrm{T}}^{\mathrm{fp}}$ p-#DirPath: This is trivial; just replace each edge of an undirected graph that is not a loop by two directed edges. Then each path (of length at least 2) in the undirected graph corresponds to exactly two paths of the same length in the directed graph. □

Next, we will prove that p-#Clique $\leq_{\mathrm{T}}^{\mathrm{fp}}$ p-#DirCycle. This requires a sequence of lemmas. Let $h : \mathcal{H} \to \mathcal{G}$ be a homomorphism and, for $i \geq 1$, let k_i be the number of vertices $b \in G$ such that $|h^{-1}(b)| \geq i$. Then $\sum_{i \geq 1} k_i = |H|$. The *type* of h is the polynomial

$$t_h(X) = \prod_{i \geq 1} (X - i + 1)^{k_i} = \prod_{b \in G} (X)_{|h^{-1}(b)|},$$

where the notation $(X)_i$ is used for the "falling factorial"; that is, $(X)_0 = 1$ and $(X)_{i+1} = (X)_i(X - i)$ for all $i \geq 0$. In particular, an embedding from $\mathcal{H}$ into $\mathcal{G}$ is a homomorphism of type $X^{|H|}$.

Let $\mathcal{D}_k$ denote the directed cycle of length k whose vertices are $1, \ldots, k$ in cyclic order. We consider the following generalization of p-#DirCycle:

p-#TDC	
Input:	Directed graph $\mathcal{G}$, polynomial $t(X)$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the homomorphisms $h : \mathcal{D}_k \to \mathcal{G}$ of type $t(X)$.

Lemma 5.3.

$$p\text{-}\#\mathrm{TDC} \leq_{\mathrm{T}}^{\mathrm{fp}} p\text{-}\#\mathrm{DirCycle}.$$

Proof. For a directed graph $\mathcal{G}$ and natural numbers $\ell, m \geq 1$, let $\mathcal{G}_{\ell,m}$ be the graph obtained from $\mathcal{G}$ as follows:

- The universe of $\mathcal{G}_{\ell,m}$ is
$$G_{\ell,m} = G \times \{1, \ldots, \ell\} \times \{1, \ldots, m\}.$$
- There is an edge from (a, i, j) to (a', i', j') in $\mathcal{G}_{\ell,m}$ either if $i = \ell$ and $i' = 1$ and there is an edge from a to a' in $\mathcal{G}$, or if $a = a'$ and $i' = i + 1$.

Figure 5.2 gives an example.

Recall that the vertices of the cycle $\mathcal{D}_k$ are $1, \ldots, k$. The *projection* of an embedding $e : \mathcal{D}_{k \cdot \ell} \to \mathcal{G}_{\ell,m}$ is the homomorphism $\pi(e) : \mathcal{D}_k \to \mathcal{G}$ which maps vertex $a \in D_k$ to the first component of $e((a - 1) \cdot \ell + 1)$; that is, we let $\pi(e)(a) = b$ if $e((a - 1) \cdot \ell + 1) = (b, i, j)$ for some $i \in \{1, \ldots, \ell\}, j \in \{1, \ldots, m\}$.

Observe that for every homomorphism $h : \mathcal{D}_k \to \mathcal{G}$ there are

$$\ell \cdot t_h(m)^\ell$$

embeddings $e : \mathcal{D}_{k \cdot \ell} \to \mathcal{G}_{\ell,m}$ with projection $\pi(e) = h$. Let T be the set of all types of homomorphisms from $\mathcal{D}_k$ into some graph. For every type $t \in T$, let x_t be the number of homomorphisms $h : \mathcal{D}_k \to \mathcal{G}$ with $t_h = t$. Then

$$b_\ell = \sum_{t \in T} x_t \cdot \ell \cdot t(m)^\ell$$

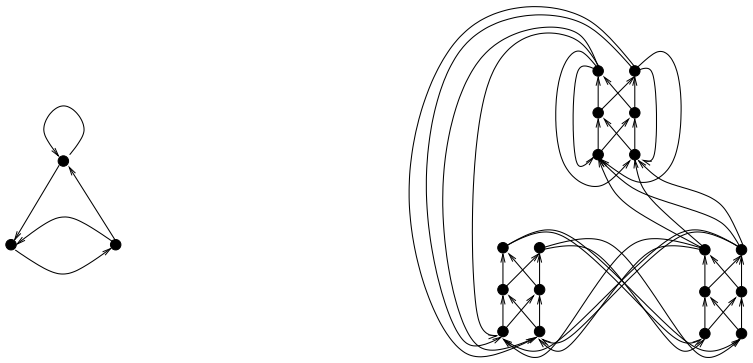


FIG. 5.2. *A directed graph $\mathcal{G}$ and the corresponding $\mathcal{G}_{3,2}$.*

is the number of embeddings $e : \mathcal{D}_{k\cdot\ell} \to \mathcal{G}_{\ell,m}$.

The types in T are polynomials of degree at most k. Thus for distinct $t(X), t'(X) \in T$ there are at most k distinct $x \in \mathbb{N}$ such that $t(x) = t'(x)$. Therefore, there is an $m \leq k \cdot |T|^2$ such that for all distinct $t(X), t'(X) \in T$ we have $t(m) \neq t'(m)$. We fix such an m.

We let $\vec{b} = (b_1, \ldots, b_{|T|})$, $\vec{x} = (x_t)_{t\in T}$, and $A = (a_{\ell t})_{\substack{1\leq \ell \leq |T| \\ t\in T}}$, where $a_{\ell t} = \ell \cdot t(m)^\ell$. Then

$$A \cdot \vec{x} = \vec{b}.$$

Since the matrix $(\frac{1}{\ell} a_{\ell t})_{\substack{1\leq \ell \leq |T| \\ t\in T}}$ is a Vandermonde matrix and thus nonsingular, the matrix A is also nonsingular, and thus

$$\vec{x} = A^{-1}\vec{b}.$$

Now our Turing reduction from p-#TDC to p-#DIRCYCLE works as follows:

1. Compute the set T and a suitable m.
2. For $1 \leq \ell \leq |T|$, compute the graph $\mathcal{G}_{\ell,m}$.
3. For $1 \leq \ell \leq |T|$, compute the number b_ℓ of embeddings $e : \mathcal{D}_{k\cdot\ell} \to \mathcal{G}_{\ell\cdot m}$ (using the oracle to p-#DIRCYCLE and noting that b_ℓ is $k \cdot \ell$ times the number of cycles of length $k \cdot \ell$ in $\mathcal{G}_{\ell\cdot m}$).
4. Compute the matrix A and solve the system $A \cdot \vec{x} = \vec{b}$.
5. Return x_t, where $t(X)$ is the input polynomial. (If $t \notin T$, then return 0.)

Since the set T, the number m, and the matrix A depend only on the parameter k, this is a parameterized Turing reduction. □

For $k, \ell \geq 1$, let $\Omega(k, \ell)$ denote the space of all mappings $f : \{1, \ldots, k \cdot \ell\} \to \{1, \ldots, k\}$ such that $|f^{-1}(i)| = \ell$ for $1 \leq i \leq k$.

LEMMA 5.4. *Let $k \geq 1$, and let $\mathcal{H} = (H, E^{\mathcal{H}})$ be a directed graph with universe $H = \{1, \ldots, k\}$ and $E^{\mathcal{H}} \neq H^2$. Then*

$$\lim_{\ell\to\infty} \Pr_{f\in\Omega(k,\ell)} \big(f \text{ is a homomorphism from } \mathcal{D}_{k\cdot\ell} \text{ to } \mathcal{H}\big) = 0$$

(where f is chosen uniformly at random).

Proof. Let $(x,y) \in H^2 \setminus E^{\mathcal{H}}$ and $m \leq k \cdot \ell$. We call a tuple $(i_1,\dots,i_m) \in H^m$ *good* if $(i_j, i_{j+1}) \neq (x,y)$ for $1 \leq j \leq m-1$ and *bad* otherwise. For $(i_1,\dots,i_m) \in H^m$ chosen uniformly at random we have

$$\begin{aligned}\Pr\big((i_1,\dots,i_m) \text{ good}\big) &\leq \Pr\big(\forall j,\, 1 \leq j \leq m/2 : (i_{2j-1}, i_{2j}) \neq (x,y)\big)\\ &= \left(1 - \frac{1}{k^2}\right)^{\lfloor m/2 \rfloor}.\end{aligned}$$

Furthermore, for all $i_1,\dots,i_m \in H$ we have

$$\Pr_{f \in \Omega(k,\ell)}\big(\forall j,\, 1 \leq j \leq m :\ f(j) = i_j\big) \leq \left(\frac{\ell}{k \cdot \ell - m}\right)^m.$$

To see this inequality, note that choosing a random function $f \in \Omega(k,\ell)$ can be modeled by randomly picking $k \cdot \ell$ balls without repetitions out of a bin that initially contains ℓ balls of each color $1,\dots,k$. The probability that the ith ball is of color j is at most

$$\frac{\ell}{k \cdot \ell - (i-1)},$$

because at most ℓ balls of the remaining $k \cdot \ell - (i-1)$ are of color j. Now the inequality follows straightforwardly.

Thus

$$\begin{aligned}&\Pr_{f \in \Omega(k,\ell)}(f \text{ is a homomorphism from } \mathcal{D}_{k\cdot\ell} \text{ to } \mathcal{H})\\ &\leq \sum_{(i_1,\dots,i_m) \in H^m \text{ good}} \Pr_{f \in \Omega(k,\ell)}(f(j) = i_j \text{ for } 1 \leq j \leq m)\\ &\leq \sum_{(i_1,\dots,i_m) \in H^m \text{ good}} \left(\frac{\ell}{k \cdot \ell - m}\right)^m\\ &= \left(\frac{\ell}{k \cdot \ell - m}\right)^m \cdot k^m \cdot \Pr_{(i_1,\dots,i_m) \in H^m}\big((i_1,\dots,i_m) \text{ good}\big)\\ &\leq \left(\frac{k \cdot \ell}{k \cdot \ell - m}\right)^m \cdot \left(1 - \frac{1}{k^2}\right)^{\lfloor m/2 \rfloor}.\end{aligned}$$

Let $\varepsilon > 0$. Then there exists an $m(\varepsilon,k)$ such that for $m \geq m(\varepsilon,k)$ we have

$$\left(1 - \frac{1}{k^2}\right)^{\lfloor m/2 \rfloor} \leq \frac{\varepsilon}{2}.$$

Moreover, for every m there exists an $\ell(m)$ such that for $\ell \geq \ell(m)$ we have

$$\left(\frac{k \cdot \ell}{k \cdot \ell - m}\right)^m = \left(\frac{1}{1 - \frac{m}{k\cdot\ell}}\right)^m \leq \frac{1}{\big(1 - \frac{m}{\ell}\big)^m} \leq 2.$$

Thus for all $\ell \geq \ell(m(\varepsilon,k))$ we have

$$\Pr_{f \in \Omega(k,\ell)}(f \text{ is a homomorphism from } \mathcal{D}_{k\cdot\ell} \text{ to } \mathcal{H}) \leq \varepsilon. \qquad \square$$

LEMMA 5.5.

$$p\text{-}\#\textsc{Clique} \leq_{\mathrm{T}}^{\mathrm{fp}} p\text{-}\#\mathrm{TDC}.$$

Proof. Let $k \geq 1$. For a graph $\mathcal{H}$, let $\overleftrightarrow{\mathcal{H}}$ denote the directed graph with the same vertex set and edge set

$$\big\{(a,a) \bigm| a \in H\big\} \cup \big\{(a,b) \bigm| \{a,b\} \in E^{\mathcal{H}}\big\}.$$

For every graph $\mathcal{H}$ with k vertices and every $\ell \geq 1$, let $a_{\mathcal{H}\ell}$ be the number of homomorphisms of type $(X)_\ell^k$ from $\mathcal{D}_{k\cdot\ell}$ into $\overleftrightarrow{\mathcal{H}}$ (that is, homomorphisms for which each point in the image has exactly ℓ preimages). Let $\overset{\mathbb{N}}{a}_{\mathcal{H}} = (a_{\mathcal{H}1}, a_{\mathcal{H}2}, \dots)$ and, for every $\ell \geq 1$, $\overset{\ell}{a}_{\mathcal{H}} = (a_{\mathcal{H}1}, a_{\mathcal{H}2}, \dots, a_{\mathcal{H}\ell})$. We consider $\overset{\mathbb{N}}{a}_{\mathcal{H}}$ and $\overset{\ell}{a}_{\mathcal{H}}$ as vectors in the vector spaces $\mathbb{Q}^{\mathbb{N}}$ and $\mathbb{Q}^\ell$, respectively.

Let $k \geq 1$, and let $\mathcal{K}$ be the complete graph with vertices $\{1,\dots,k\}$. Let $\mathbb{H}$ be the set of all graphs with vertex set $\{1,\dots,k\}$, where up to isomorphism each graph occurs only once in $\mathbb{H}$, and let $\mathbb{H}^- = \mathbb{H} \setminus \{\mathcal{K}\}$.

For a set S of vectors in $\mathbb{Q}^{\mathbb{N}}$ or $\mathbb{Q}^\ell$, we let $\langle S\rangle$ denote the linear span of S.

Claim 1.

$$\overset{\mathbb{N}}{a}_{\mathcal{K}} \notin \left\langle \left\{ \overset{\mathbb{N}}{a}_{\mathcal{H}} \bigm| \mathcal{H} \in \mathbb{H}^- \right\} \right\rangle.$$

Proof. Recall that $\Omega(k,\ell)$ denotes the set of all mappings $h : \{1,\dots,k\cdot\ell\} \to \{1,\dots,k\}$ with the property that $|h^{-1}(i)| = \ell$ for $1 \leq i \leq k$.

We first observe that for all $\ell \geq 1$,

$$a_{\mathcal{K}\ell} = |\Omega(k,\ell)|.$$

On the other hand, by Lemma 5.4 for all graphs $\mathcal{H} \in \mathbb{H}^-$ we have

$$\lim_{\ell\to\infty} \frac{a_{\mathcal{H}\ell}}{|\Omega(k,\ell)|} = 0.$$

Suppose for contradiction that

$$\overset{\mathbb{N}}{a}_{\mathcal{K}} = \sum_{i=1}^{n} \lambda_i \overset{\mathbb{N}}{a}_{\mathcal{H}_i}$$

for graphs $\mathcal{H}_1,\dots,\mathcal{H}_n \in \mathbb{H}^-$. Choose ℓ sufficiently large such that for $1 \leq i \leq n$

$$\frac{a_{\mathcal{H}_i\ell}}{a_{\mathcal{K}\ell}} = \frac{a_{\mathcal{H}_i\ell}}{|\Omega(k,\ell)|} < \frac{1}{\sum_{i=1}^n |\lambda_i|}.$$

Then

$$a_{\mathcal{K}\ell} = \sum_{i=1}^{n} \lambda_i a_{\mathcal{H}_i\ell} \leq a_{\mathcal{K}\ell} \sum_{i=1}^{n} |\lambda_i| \frac{a_{\mathcal{H}_i\ell}}{a_{\mathcal{K}\ell}} < a_{\mathcal{K}\ell} \sum_{i=1}^{n} |\lambda_i| \frac{1}{\sum_{j=1}^n |\lambda_j|} = a_{\mathcal{K}\ell},$$

which is a contradiction. This proves Claim 1.

Claim 2. There is an $\ell = \ell(k) \in \mathbb{N}$ such that

$$\overset{\ell}{a}_{\mathcal{K}} \notin \left\langle \left\{ \overset{\ell}{a}_{\mathcal{H}} \bigm| \mathcal{H} \in \mathbb{H}^- \right\} \right\rangle.$$

Furthermore, the mapping $k \mapsto \ell(k)$ is computable.

Proof. For $\iota \in \mathbb{N} \cup \{\mathbb{N}\}$, let

$$V_\iota = \left\langle \left\{ \overset{\iota}{a}_{\mathcal{H}} \;\middle|\; \mathcal{H} \in \mathbb{H}^- \right\} \right\rangle.$$

Identifying $(a_1, \dots, a_i) \in \mathbb{Q}^i$ with $(a_1, \dots, a_i, 0, 0, \dots) \in \mathbb{Q}^{\mathbb{N}}$, for all $i \geq 1$ we can view V_i as a subspace of V_j for all $j \geq i$ and of $V_{\mathbb{N}}$. Thus we can find an increasing sequence

$$\mathbb{B}_1 \subseteq \mathbb{B}_2 \subseteq \mathbb{B}_3 \subseteq \cdots \subseteq \mathbb{H}^-$$

such that for all $i \geq 1$

$$\left\{ \overset{i}{a}_{\mathcal{H}} \;\middle|\; \mathcal{H} \in \mathbb{B}_i \right\}$$

is a basis of V_i. Since $V_{\mathbb{N}}$ is a finite dimensional vector space, there is an $n \in \mathbb{N}$ such that $\mathbb{B}_i = \mathbb{B}_n$ for all $i \geq n$.

Now if $\overset{i}{a}_{\mathcal{K}}$ was in V_i for all $i \geq 1$, then for all $i \geq 1$, the vector $\overset{i}{a}_{\mathcal{K}}$ could be written as a unique linear combination of the vectors in $\left\{ \overset{i}{a}_{\mathcal{H}} \;\middle|\; \mathcal{H} \in \mathbb{B}_i \right\}$. For all $i \geq n$, these linear combinations would be identical; thus $\overset{\mathbb{N}}{a}_{\mathcal{K}}$ would be in $V_{\mathbb{N}}$. This contradicts Claim 1 and thus proves that for some $\ell \in \mathbb{N}$,

$$\overset{\ell}{a}_{\mathcal{K}} \notin V_\ell.$$

Clearly such an ℓ is computable from k, since we can compute all vectors $\overset{i}{a}_{\mathcal{H}}$ for $\mathcal{H} \in \mathbb{H}$ and $i \in \mathbb{N}$. This completes the proof of Claim 2.

Now we are ready to prove the lemma. Let $k \geq 1$ and define $\mathcal{K}$, $\mathbb{H}$, $\mathbb{H}^-$, and the vectors $\overset{\iota}{a}_{\mathcal{H}}$ as above. Choose $\ell = \ell(k)$ according to Claim 2.

Let $\mathcal{G}$ be a graph. For every graph $\mathcal{H} \in \mathbb{H}$, let $x_{\mathcal{H}}$ be the number of subsets $A \subseteq G$ such that the subgraph induced by $\mathcal{G}$ on A is isomorphic to $\mathcal{H}$. We want to determine the number $x_{\mathcal{K}}$. For $1 \leq i \leq \ell$, let b_i be the number of homomorphisms from $\mathcal{D}_{k \cdot i}$ into $\overset{\leftrightarrow}{\mathcal{G}}$ of type $(X)_i^k$, and let $\overset{\ell}{b} = (b_1, \dots, b_\ell)$. The numbers b_i can be computed by an oracle to p-#TDC.

Observe that for $1 \leq i \leq \ell$ we have

$$b_i = \sum_{\mathcal{H} \in \mathbb{H}} x_{\mathcal{H}} a_{\mathcal{H}i}$$

and thus

$$\overset{\ell}{b} = \sum_{\mathcal{H} \in \mathbb{H}} x_{\mathcal{H}} \overset{\ell}{a}_{\mathcal{H}}.$$

Since $\overset{\ell}{a}_{\mathcal{K}}$ is linearly independent from $\left\{ \overset{\ell}{a}_{\mathcal{H}} \;\middle|\; \mathcal{H} \in \mathbb{H}^- \right\}$, the coefficient $x_{\mathcal{K}}$ can be computed by solving this system of linear equations. □

Proof of Theorem 5.1. The theorem follows immediately from Lemmas 5.2, 5.3, and 5.5 and Theorem 4.4. □

6. Conclusions. We have set up a framework for a parameterized complexity theory of counting problems and proved a number of completeness results. In particular, we proved the fixed-parameter intractability of natural counting problems whose decision version is fixed-parameter tractable.

A lot of interesting problems remain open; let us just mention two of them.

- In view of Valiant's #P-completeness result for counting perfect matchings, it would be quite nice to show that the parameterized matching problem p-#Matching is #W[1]-complete. We conjecture that this is the case.
- Another interesting question is related to Toda's theorem: Does #W[1] contain the whole W-hierarchy, or maybe even the A-hierarchy (introduced in [15])?

Appendix A: A list of problems appearing in this paper.

Vertex cover and related problems. A *vertex cover* of a graph $\mathcal{G} = (G, E^{\mathcal{G}})$ is a subset $X \subseteq G$ such that for all edges $(v, w) \in E^{\mathcal{G}}$ either $v \in X$ or $w \in X$.

p-Vertex Cover	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ has a vertex cover of size k.

p-#Vertex Cover	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the vertex covers of $\mathcal{G}$ of size k.

A *dominating set* of a graph $\mathcal{G} = (G, E^{\mathcal{G}})$ is a subset $X \subseteq G$ such that for all vertices $w \in G$ either $w \in X$ or $(v, w) \in E^{\mathcal{G}}$ for some $v \in X$.

p-Dominating Set	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ has a dominating set of size k.

p-#Dominating Set	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the dominating sets of $\mathcal{G}$ of size k.

In general, the *standard parameterization* of an optimization problem is the parameterized decision problem asking whether there exists a solution of size k, where k is the quantity to be optimized and the parameter. The counting version can be defined accordingly.

Homomorphisms, embeddings, and substructures.

p-Hom	
Input:	Structures $\mathcal{A}$ and $\mathcal{B}$.
Parameter:	$\|\mathcal{A}\|$.
Problem:	Decide if there exists a homomorphism from $\mathcal{A}$ to $\mathcal{B}$.

p-#Hom	
Input:	Structures $\mathcal{A}$ and $\mathcal{B}$.
Parameter:	$\|\mathcal{A}\|$.
Problem:	Count the homomorphisms from $\mathcal{A}$ to $\mathcal{B}$.

p-Emb	
Input:	Structures $\mathcal{A}$ and $\mathcal{B}$.
Parameter:	$\|\mathcal{A}\|$.
Problem:	Decide if there exists an embedding of $\mathcal{A}$ into $\mathcal{B}$.

p-#Emb	
Input:	Structures $\mathcal{A}$ and $\mathcal{B}$.
Parameter:	$\|\mathcal{A}\|$.
Problem:	Count the embeddings of $\mathcal{A}$ into $\mathcal{B}$.

p-Sub	
Input:	Structures $\mathcal{A}$ and $\mathcal{B}$.
Parameter:	$\|\mathcal{A}\|$.
Problem:	Decide if $\mathcal{B}$ has a substructure isomorphic to $\mathcal{A}$.

p-#Sub	
Input:	Structures $\mathcal{A}$ and $\mathcal{B}$.
Parameter:	$\|\mathcal{A}\|$.
Problem:	Count the substructures of $\mathcal{B}$ isomorphic to $\mathcal{A}$.

A *clique* in a graph $\mathcal{G}$ is a subset X of G such that for all distinct $v, w \in X$, $(v, w) \in E^{\mathcal{G}}$.

p-Clique	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ has a clique of size k.

p-#Clique	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the cliques of $\mathcal{G}$ of size k.

p-Path	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ contains a path of length k.

p-#Path	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the paths of length k in $\mathcal{G}$.

p-DirPath	
Input:	Directed graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ contains a directed path of length k.

p-#DirPath	
Input:	Directed graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the directed paths of length k in $\mathcal{G}$.

p-Cycle	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ contains a cycle of length k.

p-#Cycle	
Input:	Graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the cycles of length k in $\mathcal{G}$.

p-DirCycle	
Input:	Directed graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ contains a directed cycle of length k.

p-#DirCycle	
Input:	Directed graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the directed cycles of length k in $\mathcal{G}$.

A *matching* of a graph is a set of edges that pairwise have no endpoint in common.

p-Matching	
Input:	Bipartite graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ contains a matching of size k.

p-#Matching	
Input:	Bipartite graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the matchings of size k in $\mathcal{G}$.

Logically defined problems. The *weight* of an assignment S for the variables of a propositional formula is the number of variables set to TRUE by S. Let Θ be a class of propositional formulas

WSAT(Θ)	
Input:	$\theta \in \Theta$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if θ has a satisfying assignment of weight k.

#WSAT(Θ)	
Input:	$\theta \in \Theta$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the satisfying assignments of θ of weight k.

$|\varphi|$ denotes the length of a formula φ. Let Φ be a class of first-order formulas.

p-MC(Φ)	
Input:	Structure $\mathcal{A}$, formula $\varphi \in \Phi$.
Parameter:	$\lvert\varphi\rvert$.
Problem:	Decide if $\varphi(\mathcal{A}) \neq \emptyset$.

p-#(Φ)	
Input:	Structure $\mathcal{A}$, formula $\varphi \in \Phi$.
Parameter:	$\lvert\varphi\rvert$.
Problem:	Compute $\lvert\varphi(\mathcal{A})\rvert$.

Let $\varphi(X)$ be a formula of vocabulary $\tau \cup \{X\}$.

p-FD($\varphi(X)$)	
Input:	τ-structure $\mathcal{A}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\varphi(\mathcal{A})$ contains a relation of size k.

p-#FD($\varphi(X)$)	
Input:	Structure $\mathcal{A}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the number of relations of size k in $\varphi(\mathcal{A})$.

The parameterized halting problem.

p-HALT	
Input:	Nondeterministic Turing machine M.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if M accepts the empty word in at most k steps.

p-#HALT	
Input:	Nondeterministic Turing machine M.
Parameter:	$k \in \mathbb{N}$.
Problem:	Count the accepting computation paths of M of length at most k for the empty word.

REFERENCES

[1] K. A. Abrahamson, R. G. Downey, and M. R. Fellows, *Fixed-parameter tractability and completeness.* IV. *On completeness for W[P] and PSPACE analogues*, Ann. Pure Appl. Logic, 73 (1995), pp. 235–276.

[2] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. ACM, 42 (1995), pp. 844–856.

[3] N. Alon, R. Yuster, and U. Zwick, *Finding and counting given length cycles*, Algorithmica, 17 (1997), pp. 209–223.

[4] S. Arnborg, J. Lagergren, and D. Seese, *Easy problems for tree-decomposable graphs*, J. Algorithms, 12 (1991), pp. 308–340.

[5] V. Arvind and V. Raman, *Approximation algorithms for some parameterized counting problems*, in Proceedings of the 13th Annual International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 2518, P. Bose and P. Morin, eds., Springer-Verlag, Berlin, 2002, pp. 453–464.

[6] H. L. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett, and H. T. Wareham, *Parameterized complexity analysis in computational biology*, CABIOS, 11 (1995), pp. 49–57.

[7] L. Cai and J. Chen, *On fixed-parameter tractability and approximability of NP optimization problems*, J. Comput. System Sci., 54 (1997), pp. 465–474.

[8] Y. Chen, J. Flum, and M. Grohe, *Bounded nondeterminism and alternation in parameterized complexity theory*, in Proceedings of the 18th IEEE Conference on Computational Complexity, 2003, pp. 13–29.

[9] B. Courcelle, *Graph rewriting: An algebraic and logic approach*, in Handbook of Theoretical Computer Science, Vol. B, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 193–242.

[10] B. Courcelle, J. A. Makowsky, and U. Rotics, *On the fixed-parameter complexity of graph enumeration problems definable in monadic second-order logic*, Discrete Appl. Math., 108 (2001), pp. 23–52.

[11] R. G. Downey and M. R. Fellows, *Fixed-parameter tractability and completeness* I: *Basic results*, SIAM J. Comput., 24 (1995), pp. 873–921.

[12] R. G. Downey and M. R. Fellows, *Fixed-parameter tractability and completeness* II: *On completeness for $W[1]$*, Theoret. Comput. Sci., 141 (1995), pp. 109–131.

[13] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999.

[14] R. G. Downey, M. R. Fellows, and K. W. Regan, *Descriptive complexity and the W hierarchy*, in Proof Complexity and Feasible Arithmetics, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 39, P. Beame and S. Buss, eds., AMS, Providence, RI, 1998, pp. 119–134.

[15] J. Flum and M. Grohe, *Fixed-parameter tractability, definability, and model-checking*, SIAM J. Comput., 31 (2001), pp. 113–145.

[16] M. Frick, *Generalized model-checking over locally tree-decomposable classes*, in Proceedings of the 19th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2285, H. Alt and A. Ferreira, eds., Springer-Verlag, New York, 2002, pp. 632–644.

[17] M. Frick and M. Grohe, *Deciding first-order properties of locally tree-decomposable structures*, J. ACM, 48 (2001), pp. 1184–1206.

[18] G. Gottlob, N. Leone, and M. Sideri, *Fixed-parameter complexity in AI and nonmonotonic reasoning*, in Logic Programming and Nonmonotonic Reasoning, 5th International Conference, LPNMR'99, Lecture Notes in Comput. Sci. 1730, M. Gelfond, N. Leone, and G. Pfeifer, eds., Springer-Verlag, Berlin, 1999, pp. 1–18.

[19] M. Grohe, *The parameterized complexity of database queries*, in Proceedings of the 20th ACM Symposium on Principles of Database Systems, 2001, ACM Press, New York, pp. 82–92.

[20] M. Jerrum, A. Sinclair, and E. Vigoda, *A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries*, in Proceedings of the 33rd ACM Symposium on Theory of Computing, 2001, ACM Press, New York, pp. 712–721.

[21] Ph. G. Kolaitis and M. N. Thakur, *Approximation properties of NP minimization classes*, J. Comput. System Sci., 50 (1995), pp. 391–411.

[22] J. A. Makowsky, *Colored Tutte polynomials and Kauffman brackets for graphs of bounded tree width*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 487–495.

[23] C. McCartin, *Parameterized counting problems*, in Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 2420, K. Diks and W. Rytter, eds., Springer-Verlag, Heidelberg, 2002, pp. 556–567.

[24] C. H. Papadimitriou and M. Yannakakis, *On the complexity of database queries*, in Proceedings of the 17th ACM Symposium on Principles of Database Systems, 1997, ACM Press, New York, pp. 12–19.

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

[26] J. Plehn and B. Voigt, *Finding minimally weighted subgraphs*, in Graph-Theoretic Concepts in Computer Science, Lecture Notes in Comput. Sci. 484, R. Möhring, ed., Springer-Verlag, Berlin, 1991, pp. 18–29.

[27] U. Stege, *Resolving Conflicts in Problems from Computational Biology*, Ph.D. thesis 13364, ETH Zürich, Zürich, Switzerland, 2000.

[28] S. Toda, *PP is as hard as the polynomial-time hierarchy*, SIAM J. Comput., 20 (1991), pp. 865–877.

[29] L. G. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 923–936

ON WORST-CASE ROBIN HOOD HASHING*

LUC DEVROYE†, PAT MORIN†, AND ALFREDO VIOLA‡

Abstract. We consider open addressing hashing and implement it by using the Robin Hood strategy; that is, in case of collision, the element that has traveled the farthest can stay in the slot. We hash $\sim \alpha n$ elements into a table of size n where each probe is independent and uniformly distributed over the table, and $\alpha < 1$ is a constant. Let M_n be the maximum search time for any of the elements in the table. We show that with probability tending to one, $M_n \in [\log_2 \log n + \sigma, \log_2 \log n + \tau]$ for some constants σ, τ depending upon α only. This is an exponential improvement over the maximum search time in case of the standard FCFS (first come first served) collision strategy and virtually matches the performance of multiple-choice hash methods.

Key words. open addressing, hashing, Robin Hood, worst-case search time, collision resolution, probabilistic analysis of algorithms

AMS subject classifications. 60D05, 68U05

DOI. 10.1137/S0097539702403372

1. Introduction.

In hashing with chaining with a table of size n holding $m = \lceil \alpha n \rceil$ elements, where $\alpha > 0$ is a constant, the worst-case search time is equal to the length of the longest chain. If the hash values are independent and uniformly distributed over the table, then the maximum chain length is asymptotic to $\log n / \log \log n$ in probability (Gonnet (1981); Devroye (1985)) for any fixed value of α.

In this paper we consider open addressing hashing with random probing. A table of size n is given, into which we place $m = \lceil \alpha n \rceil$ elements, where $\alpha \in (0, 1)$ is a fixed constant. Each element has associated with it an infinite probe sequence consisting of independently and identically distributed (i.i.d.) integers uniformly distributed over $\{1, \ldots, n\}$, representing the consecutive places of probes for that element. It is assumed that when searching for an element, its infinite probe sequence is available to the searcher. The probe sequence for the ith element is denoted by $X_{i,0}, X_{i,1}, X_{i,2}, \ldots$. Elements are inserted sequentially into the table. If the ith element is placed in position $X_{i,j}$, then we say that the ith element has age j, as it requires j hops to reach the element in case of a search. When the ith element of age j and the i'th element of age j' compete for the same slot ($X_{i,j} = X_{i',j'}$), a collision resolution strategy is needed. Several collision resolution strategies have dominated the literature.

The standard open addressing method resolves the collision by giving the place to the first key to arrive there according to a first come first served (FCFS) policy, so the test is based on $\min(i, i')$. Amble and Knuth (1974) suggested the idea that *any* of the colliding elements could get the position in the hope of speeding up unsuccessful searches. Note that for random probing, for any strategy that does not look ahead, the sum of the ages of all elements in a hash table has a distribution that is independent

*Received by the editors February 28, 2002; accepted for publication (in revised form) September 29, 2003; published electronically May 25, 2004.
http://www.siam.org/journals/sicomp/33-4/40337.html

†School of Computer Science, McGill University, Montreal, Canada H3A 2K6 (luc@cs.mcgill.ca, morin@scs.carleton.ca). Research for these authors was supported by NSERC grant A3456 and FCAR grant 90-ER-0291.

‡Pedeciba Informatica, Districto 6, Casilla de Correo 16120, Universidad de la República, Montevideo, Uruguay (viola@fing.edu.uy). Research for this author was supported by Proyectos de investigación CSIC fondos 2000-2002 and 2002-2004 at Universidad de la República.

of the collision resolution strategy. There are differences, though, when one considers the maximal age among all elements in a table. Two of the strategies that do not look ahead before deciding which element should get the position are the LCFS (last come first served) heuristic (Poblete and Munro (1989)), in which the position is given to the last element that arrives (thus, using $\max(i, i')$), and the Robin Hood strategy (Celis (1986); Celis, Larson, and Munro (1985); Viola and Poblete (1998)), in which the position is given to the element that is farthest away from its home location (the element corresponding to $\max(j, j')$). The Robin Hood strategy tends to equalize the ages of all inserted elements (hence the name Robin Hood), thus reducing the maximum successful search time. Both FCFS and Robin Hood decrease the variance of the search time. As pointed out earlier, for random probing, the expected search time for a single random element is identical for all collision resolution strategies that do not look ahead. An interesting property of Robin Hood is that every permutation of the insertion sequence produces the same final hash table, provided that a consistent tiebreaker is used (for example, $\min(i, i')$).

In open addressing hashing, most of the proposed schemes to improve the search cost of a random element in a hash table (like Brent's method, binary tree, optimal and min-max hashing) have very high cost for table creation. Other methods like multiple-choice hashing are more inefficient in the use of space. As presented in Celis (1986), Robin Hood is an open addressing hashing scheme that is as simple to program as the standard algorithm, takes only $\Theta(n \log n)$ on the average to load a full table, requires no additional memory for insertions, and has very small variance. This last fact is a key observation in Celis (1986), to speed up the searching cost of a random element. The main idea is not to probe the first position in the probe sequence but rather the most probable place and then move away from it in both directions.

For uniform probing (that is, a probe sequence without repetition) the expected value of the longest probe sequence for the standard FCFS algorithm for α-full tables ($\alpha < 1$) is $\log_{1/\alpha} n - \log_{1/\alpha}(\log_{1/\alpha} n) + O(1)$ and for full tables is $0.631587\ldots \times n + O(1)$ (Gonnet (1981)).

Poblete and Munro (1989) prove that for random probing (that is, a probe sequence with repetition) the expected value of the longest probe sequence for the LCFS heuristic is bounded by

$$1 + \Gamma^{-1}(\alpha n)\left(1 + \frac{\log\log(1/(1-\alpha))}{\log \Gamma^{-1}(\alpha n)} + O\left(\frac{1}{\log^2 \Gamma^{-1}(\alpha n)}\right)\right),$$

where Γ is the Gamma function, and

$$\Gamma^{-1}(\alpha n) = \frac{\log n}{\log\log n}\left(1 + \frac{\log\log\log n}{\log\log n} + O\left(\frac{1}{\log\log n}\right)\right).$$

Although this is not a tight bound, this was the first open addressing method for which a sublogarithmic bound in n was proven.

Celis (1986) proves that the expected value of the longest probe sequence for random probing and a full Robin Hood hash table ($\alpha = 1$) is $\Theta(\log n)$. Moreover, when $\alpha < 1$ he proved that for random probing, the expected value of the longest probe sequence for the Robin Hood heuristic is bounded by $3(H_n - H_{n-m})/\alpha + \lceil \log(n-2) \rceil$, where $H_n = \sum_{1 \le i \le n} 1/i$. This bound is improved in this paper to $\log_2 \log n$. For further discussions and results, see Knuth (1998), Vitter and Flajolet (1990), Gonnet and Baeza-Yates (1991), or Flajolet, Poblete, and Viola (1998). It is perhaps worth

TABLE 1.1
Expected length of longest successful probe sequence.

n	$\alpha = 0.6$	$\alpha = 0.7$	$\alpha = 0.8$	$\alpha = 0.9$	$\alpha = 1.0$
1021	3.629 ± .065	4.000 ± .013	4.329 ± .064	5.105 ± .041	10.443 ± .187
4093	3.967 ± .024	4.062 ± .033	4.800 ± .054	5.329 ± .064	12.133 ± .208
16273	4.014 ± .016	4.262 ± .060	5.000 ± .000	5.771 ± .057	13.819 ± .172
65537	4.029 ± .023	4.614 ± .066	5.000 ± .000	6.000 ± .000	15.181 ± .178
262139	4.098 ± .040	4.967 ± .024	5.022 ± .020	6.000 ± .000	16.815 ± .179

reproducing Table 5.9 from Celis's dissertation, in which empirical estimates were computed for the longest successful probe length with the Robin Hood strategy, which suggests a $\Theta(\log\log n)$ complexity for the problem when $\alpha < 1$ (see Table 1.1).

It is perhaps worth mentioning that there are several other ways of obtaining dynamic hash tables with $O(\log\log n)$ expected maximum successful search times. Consider hashing with chaining, and let the elements have a choice of two randomly picked positions. An element is placed into the slot with the least number of elements (at the time of insertion). This simple double choice shows that the maximum slot occupancy is in probability asymptotic to $\log_2\log_2 n$ (Azar et al. (1999); Broder and Karlin (1990); Czumaj and Stemann (1997); Mitzenmacher (1997)).

There has been interest in obtaining $O(1)$ expected worst-case performance, or even $O(1)$ deterministic worst-case performance, for search in hash tables. For static hash tables, Fredman, Komlós, and Szemerédi (1984) proposed a solution. Czumaj and Stemann (1997) showed that if each element has two randomly chosen hash positions, then with high probability, a static (off-line) chaining hash table can be constructed that has worst chain length 2, provided that the table size is at least αn for some threshold constant α. For dynamic hash tables, the early research was in the direction of dynamic perfect hash functions (Dietzfelbinger and Meyer auf der Heide (1990); Dietzfelbinger et al. (1992); Dietzfelbinger et al. (1994); Brodnik and Munro (1999)). Cuckoo hashing (Pagh and Rodler (2001)) is also an attempt in this direction. It stands out though through its simplicity and the promising experimental results reported by Pagh and Rodler: each of m data points has two hash functions, one to be used in each of two tables of size $n \geq (1+\epsilon)m$. The element must be placed in one of the tables at one of the two locations. Upon insertion of a new element, old elements get kicked out and move around, kicking out other elements if necessary, until either a loop is detected or the insertion process halts. In case of a loop, the entire table is rehashed. The expected time for an insertion is still $O(1)$, and the worst-case successful search time is bounded by 2. However, one needs a powerful collection of hash functions, as each rehash operation requires an entirely new and independent set of hash values.

Let us denote by M_n the maximal successful search time, that is, the maximal age among any of the m elements in the hash table, and by T_n the maximum insertion cost of an element. In an FCFS strategy, we note that $M_n = T_n - 1$, but this is no longer true for other strategies. In fact, in this paper we show the following.

THEOREM 1.1. *In open addressing with Robin Hood collision resolution, there exists a constant C depending upon α only such that*

$$\lim_{n\to\infty} \mathbf{P}\{M_n \geq \log_2\log n + C\} = 0 \ .$$

We will see that $C \to \infty$ when $\alpha \to 1$, so this result is meaningful only when

$\alpha < 1$. The result above implies an exponential improvement over the FCFS strategy. Furthermore, this bound is optimal modulo a finite constant, as follows.

THEOREM 1.2. *In open addressing with Robin Hood collision resolution (and any method of breaking ties),*

$$\mathbf{P}\{M_n \le \log_2 \log n - \log_2(6\log(8/\alpha))\} = O(1/\sqrt{n}) \ .$$

The implications of this should not be underestimated, as open addressing tables are the oldest and simplest hashing structures. The multiple-choice hashing methods in their original form are intrinsically chaining methods, and thus slightly more inefficient spacewise.

The $\log_2 \log n$ behavior follows, roughly speaking, from the following observation. If we place all m elements in their first choice bins, then all but one element from each bin must move to another bin. The number of these excess elements is about m^2/n times a constant. Just looking at these displaced elements, we repeat the argument k times, obtaining increasingly smallest sets to be displaced. After k steps, the number of elements left is of the order of $n(m/n)^{2^k}$, or $n\alpha^{2^k}$. This is of constant order when k is about $\log_2 \log n$.

1.1. Balls in urns. Throw m balls uniformly at random into n urns. Let urn i receive N_i balls, and define

$$A = \sum_{i=1}^{n} (N_i - 1)_+$$

as the number of balls left after removing one ball from each occupied urn. We say that A has the (m, n) urn distribution.

The (m, n) urn distribution. Let A have the (m, n) urn distribution. Then

$$\mathbf{E}\{A\} = \sum_{j=1}^{m} \left(1 - (1 - 1/n)^{j-1}\right) \ .$$

Note that $(1 - 1/n)^m \ge 1 - m/n$ and $(1 - 1/n)^m \le 1 - m/n + m(m-1)/2n^2$, so that

$$\begin{aligned}
\frac{m^2}{2n} &\ge \frac{m(m-1)}{2n} \\
&= \sum_{j=1}^{m-1} \frac{j}{n} \\
&\ge \mathbf{E}\{A\} \\
&\ge \sum_{j=1}^{m-1} \frac{j}{n} - \sum_{j=1}^{m-1} \frac{j(j-1)}{2n^2} \qquad (1) \\
&= \frac{m(m-1)}{2n} - \frac{m(m-1)(m-2)}{6n^2} \\
&\ge \frac{m(m-1)}{3n} \\
&\ge \frac{m^2}{4n} \quad \text{(the last step is true only if } m \ge 4) \ .
\end{aligned}$$

We also need some concentration inequalities for A. To present these inequalities, let $(X_1, \ldots, X_n)$ be a vector of independent random variables (on an arbitrary measurable space S), let $f : S \to \mathbf{R}$ be a measurable function, and set

$$Z = f(X_1, \ldots, X_m) \ .$$

Let $X_1', \ldots, X_m'$ be independent copies of $X_1, \ldots, X_m$, and write

$$Z^{(i)} = f(X_1, \ldots, X_{i-1}, X_i', X_{i+1}, \ldots, X_m) \ .$$

The Efron–Stein inequality (Efron and Stein (1981); Steele (1986)) states that

$$\mathbf{V}\{Z\} \le \frac{1}{2}\mathbf{E}\left\{\sum_{i=1}^{m}(Z - Z^{(i)})^2\right\} \ .$$

If $Z \equiv A$, and $X_1, \ldots, X_m$ are the urns chosen by elements 1 through m, and X_i' is independent of the X_j's and distributed as X_i, then $|Z^{(i)} - Z| \le 1$. Thus, $\mathbf{V}\{Z\} \le m/2$. With the inequalities for $\mathbf{E}\{A\}$ taken into account, we have, by Chebyshev's inequality, for all $t > 0$,

$$\mathbf{P}\left\{|A - \mathbf{E}\{A\}| \ge t\right\} \le \frac{\mathbf{V}\{A\}}{t^2} \le \frac{m}{2t^2} \ .$$

1.2. The head-and-belly view. The construction of the hash table may be looked at in a global manner for Robin Hood strategies, since every permutation of the input sequence produces the same hash table. We start by placing all elements at their first choices $X_{i,0}$, $1 \le i \le m$. Some bins in the table may have many elements, but that is acceptable. We call this the first stage. At the kth stage in our construction, picture a hash table ("the head") containing elements of age k, possibly many per cell, and a second hash table ("the belly") containing at most one element per cell, and that element is of age less than k. Furthermore—and this is crucial—if cell i in the head is occupied, then cell i is empty in the belly. This head-and-belly view allows us to proceed, by letting k grow until finally the head is empty, and all elements are in the belly.

The belly is initially empty, and all elements are in the head, in stage one. Given the $(k-1)$st stage situation, we construct the kth stage as follows (see Figure 1.1).

A. All elements in the $(k-1)$-stage head that are in positions two and above in their bins move to a randomly selected bin in the k-head.

B. The remaining elements of the $(k-1)$-head (at most one per cell) are added to the $(k-1)$-belly (in the corresponding position). Note that this may create some conflicts with the k-head just created.

C. While there is a head-belly conflict, take a conflicting element in the belly (that is, an element in cell i, such that the k-head also has an element in cell i), and let it start hopping uniformly and randomly (and aging by one with each hop), according to the rules of Robin Hood hashing, until it, or the element it causes to move, finds a position in a cell by itself in the belly, without conflict with the k-head, or a position in the k-head (an element that reaches age k must move to the k-head). In the latter case, a new conflict may be triggered. At the end of this, there is no further conflict, and the resulting tables are called the k-head and the k-belly.

LEMMA 1.3. *Let N be the number of elements added to the k-head in step C, given that the k-head has at most K elements to start with after steps A and B,*

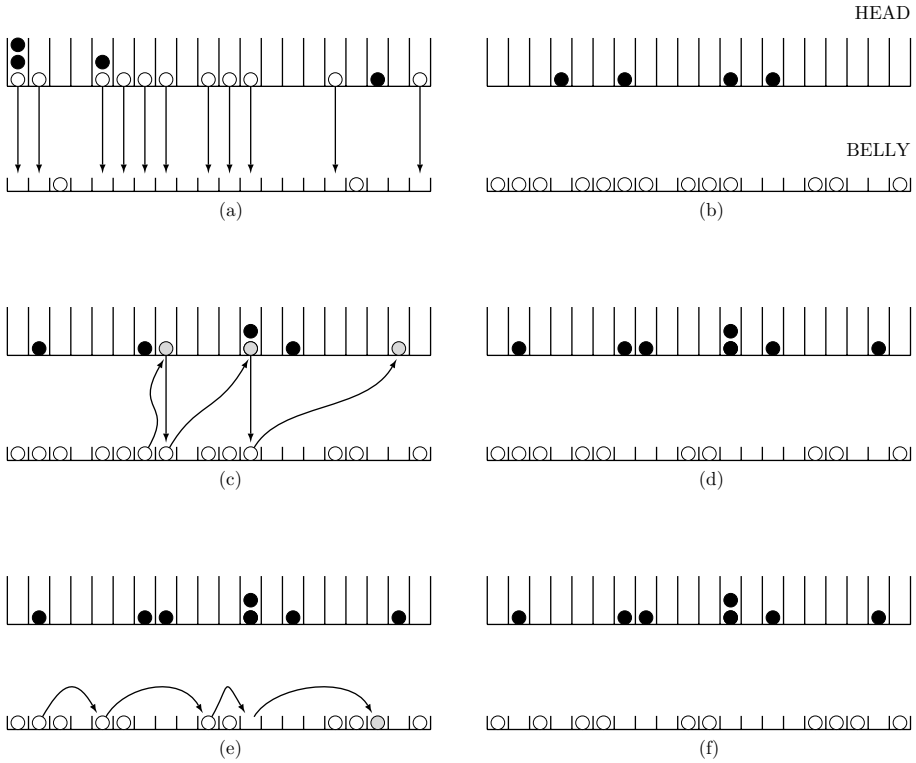


FIG. 1.1. *In* (a)*, we show a* $(k-1)$*-head that is not empty. The elements in position one in their bins (in white) move to the belly (step B). The other elements (in black) move to a random position in the* k*-head, shown in figure* (b)*. This is step A. Clearly, there are some conflicts between head and belly in* (b)*. In step C, these are resolved. For each conflict, an element in the belly is taken and is moved to a random position in the belly. For example, in* (c)*, we show the moves of an element, as it first ages to age* k *(so that its randomly picked position lands it in the head), which triggers a new conflict in the belly, which is immediately taken care of by letting that element move to a random position, which again happens to be in the head (gray element), and finally, the last conflict generated leads to yet another element in the head, causing no further conflicts. The resulting configuration is* (d)*. In* (e)*, the last remaining conflict is taken care of by random hops, resulting in the final configuration* (f) *of the* k*-belly and* k*-head. In example* (e)*, all hops remain in the belly, and result finally in a cell in the belly being filled with a new element.*

and given any distribution of elements in belly and head at that point. Then, with $\lambda = 1/\log(1/\alpha)$*,* N *is stochastically smaller than* $\lambda G_K + K$*, where* G_K *is a gamma* (K) *random variable. In particular,* $\mathbf{E}\{N\} \le (\lambda+1)K$*, and*

$$\mathbf{P}\{N \ge (2\lambda+1)K\} \le \left(\frac{2}{e}\right)^K .$$

Proof. There are initially at most K elements in the belly that can cause a conflict with the head. When these elements move, at each step we have a probability at least $1-\alpha$ of finding an empty slot (empty for both head and belly). When such a slot is found, the chain of moves ends. In each step, at most one element moves to the k-head. The number of additions to the k-head to just eliminate one belly conflict is thus stochastically smaller than one plus a geometric (α) random variable Y:

$$\mathbf{P}\{Y \ge i\} \le \alpha^i, \quad i \ge 0 .$$

If E is unit exponential, then we see that

$$\mathbf{P}\{\lambda E \geq i\} = \exp\left(-\frac{i}{\lambda}\right) = \alpha^i \ ,$$

provided that $\alpha = \exp(-1/\lambda)$ or $\lambda = 1/\log(1/\alpha)$. Therefore, $Y \prec E/\log(1/\alpha)$. Since we have to eliminate K possible conflict elements in the belly, the total number of elements added to the head is stochastically smaller than $K + Y_1 + \cdots + Y_K$, where the Y_i's are independent and are all stochastically dominated by λE. Thus, if $E_1, \ldots, E_K$ are independent exponential random variables, and G_K is a gamma (K) random variable, we see that the number N of additions to the head in part C is stochastically smaller than $K + \lambda(E_1 + \cdots + E_K) \stackrel{\mathcal{L}}{=} K + \lambda G_K$. In other words,

$$\begin{aligned}
\mathbf{P}\{N \geq (2\lambda+1)K\} &\leq \mathbf{P}\{\lambda G_K \geq 2\lambda K\} \\
&= \mathbf{P}\{G_K \geq 2K\} \\
&\leq \mathbf{E}\left\{e^{tG_K} e^{-2tK}\right\} \quad (\text{any } t > 0) \\
&= \left(\frac{e^{-2t}}{1-t}\right)^K \\
&= \left(\frac{2}{e}\right)^K \quad (\text{take } t = 1/2) \ .
\end{aligned}$$

This concludes the proof of Lemma 1.3. □

It is important to note that if $\alpha \to 1$, then $\lambda \to \infty$, so the results below are meaningful only when $\alpha < 1$.

LEMMA 1.4. *Define $b = (2\lambda+2)\alpha$ and assume that $b < 1$. Let D be the integer*

$$D = \left\lfloor \log_2\left(\frac{2}{3\log(1/b)}\right) - 0.1 \right\rfloor \ .$$

Let Z be the number of elements in the r-head, with $r = \lfloor \log_2 \log n \rfloor + D$. Then

$$\lim_{n\to\infty} \mathbf{P}\left\{Z \geq \frac{nb^{2^r}}{2\lambda+2}\right\} = 0 \ .$$

In particular,

$$\lim_{n\to\infty} \mathbf{P}\left\{Z \geq n^{1-1/(6\times 2^{0.1})}\right\} = 0 \ .$$

Proof. Given that the $(k-1)$-head has K elements or less, then if A denotes the number of elements in the k-head after step A (not including steps B and C), we have

$$\mathbf{E}\{A\} \leq \frac{K^2}{2n}$$

and

$$\mathbf{P}\{|A - \mathbf{E}\{A\}| \geq t\} \leq \frac{K}{2t^2} \ .$$

In particular, we note that

$$\mathbf{P}\left\{A \geq \frac{K^2}{n}\right\} \leq \frac{2n^2}{K^3} \ .$$

After steps B and C, N more elements are added to the k-head. We have

$$\mathbf{P}\left\{A+N \geq \frac{(2\lambda+2)K^2}{n}\right\} \leq \mathbf{P}\left\{A \geq \frac{K^2}{n}\right\} + \mathbf{P}\left\{N \geq \frac{(2\lambda+1)K^2}{n} \Big| A \leq \frac{K^2}{n}\right\}$$
$$\leq \frac{2n^2}{K^3} + \left(\frac{2}{e}\right)^{\frac{K^2}{n}} .$$

Now define the sequence a_k by $a_0 = m$,

$$a_{k+1} = \frac{(2\lambda+2)a_k^2}{n} .$$

Then it is easy to see that for $k > 0$,

$$a_k = \frac{n}{2\lambda+2}\left(\frac{(2\lambda+2)a_0}{n}\right)^{2^k} = \frac{n}{2\lambda+2}\left((2\lambda+2)\alpha\right)^{2^k} .$$

Let A_k, N_k denote the k-head cardinalities as defined above. Then

$$\begin{aligned}\mathbf{P}\{A_r + N_r \geq a_r\} \leq\ & \mathbf{P}\left\{A_r + N_r \geq a_r \mid A_{r-1} + N_{r-1} \leq a_{r-1}\right\} \\ & + \mathbf{P}\{A_{r-1} + N_{r-1} \geq a_{r-1} \mid A_{r-2} + N_{r-2} \leq a_{r-2}\} \\ & + \cdots + \mathbf{P}\{A_1 + N_1 \geq a_1 \mid A_0 + N_0 \leq a_0\} ,\end{aligned}$$

since $A_0 + N_0 = m = a_0$. By the definition of the a_k sequence, we note that the general term

$$\mathbf{P}\{A_k + N_k \geq a_k \mid A_{k-1} + N_{k-1} \leq a_{k-1}\}$$

is bounded by

$$\frac{2n^2}{a_{k-1}^3} + \left(\frac{2}{e}\right)^{\frac{a_{k-1}^2}{n}} .$$

Thus, defining $b = (2\lambda+2)\alpha$, and assuming that $b < 1$, we have

$$\begin{aligned}\mathbf{P}\{A_r + N_r \geq a_r\} &\leq \sum_{k=0}^{r-1}\left(\frac{2n^2}{a_k^3} + \left(\frac{2}{e}\right)^{\frac{a_k^2}{n}}\right) \\ &= \sum_{k=0}^{r-1}\left(\frac{2(2\lambda+2)^3}{n\left((2\lambda+2)\alpha\right)^{3\times 2^k}} + \left(\frac{2}{e}\right)^{\frac{\left((2\lambda+2)\alpha\right)^{2^{k+1}} n}{(2\lambda+2)^2}}\right) \\ &\leq C\left(\frac{1}{nb^{3\times 2^{r-1}}} + \left(\frac{2}{e}\right)^{\frac{nb^{2^r}}{(2\lambda+2)^2}}\right) \\ &\qquad \text{(for some constant } C) .\end{aligned}$$

Let $r = \lfloor \log_2 \log n \rfloor + D$ for some integer D. Then $2^{D-1}\log n \leq 2^r \leq 2^D \log n$, and $nb^{2^r} \geq nb^{3\times 2^{r-1}} \geq nb^{(3/2)2^D \log n} = n^{1+(3/2)2^D \log b}$. Thus, if $2^D \log(1/b) < 2/3$, then

$$\lim_{n\to\infty} \mathbf{P}\{A_r + N_r \geq a_r\} = 0 .$$

The last statement follows from the fact that

$$\begin{aligned} nb^{2^r} &\le nb^{2^{D-1}\log n} \\ &= n^{1-2^{D-1}\log(1/b)} \\ &\le n^{1-2^{\log_2\left(\frac{2}{3\log(1/b)}\right)-2.1}\log(1/b)} \\ &= n^{1-1/(6\times 2^{0.1})} \ . \end{aligned}$$

This concludes the proof of Lemma 1.4. □

Remark. The condition $b = (2\lambda+2)\alpha < 1$ reduces to $(2 + 2/\log(1/\alpha))\alpha < 1$. This is satisfied if $\alpha \le 0.306891\ldots$.

LEMMA 1.5. *Let r be as in Lemma 1.4. Then the probability that the $(r+3)$-head has at least one element is $o(1)$. Thus, with probability tending to one, the maximum successful search time is bounded by $r+2$.*

Proof. Let r be as in Lemma 1.4, and let Z be the number of elements in the r-head. Then it is of interest to study Z_j, the number of elements in the $(r+j)$-head, for $j > 0$. Recall that $a_r \le n^{1-1/(6\times 2^{0.1})}$. Given Z, we have $\mathbf{E}\{Z_1 \mid Z\} \le (2+\lambda)Z^2/2n$, where we used (1) and Lemma 1.3. On $Z \le a_r$, we have $\mathbf{E}\{Z_1 \mid Z\} \le (2+\lambda)a_r^2/2n \le (2+\lambda)n^{1-2^{0.9}/6}$. Thus, $\mathbf{P}\{Z_1 > (2+\lambda)\log n \times n^{1-2^{0.9}/6} \mid Z\} \le 1/\log n$ by Markov's inequality, on $Z \le a_r$. Next, on $Z_1 \le (2+\lambda)\log n \times n^{1-2^{0.9}/6}$,

$$\mathbf{E}\{Z_2 \mid Z_1\} \le (2+\lambda)\left((2+\lambda)\log n \times n^{1-2^{0.9}/6}\right)^2/2n < (2+\lambda)^3\log^2 n \times n^{1-2^{1.9}/6} \ .$$

Thus,

$$\mathbf{P}\left\{Z_2 > (2+\lambda)^3\log^3 n \times n^{1-2^{1.9}/6} \mid Z_1\right\} \le \frac{1}{\log n} \ .$$

Finally, on $Z_2 \le (2+\lambda)^3\log^3 n \times n^{1-2^{1.9}/6}$,

$$\mathbf{E}\{Z_3 \mid Z_2\} \le (2+\lambda)\left((2+\lambda)^3\log^3 n \times n^{1-2^{1.9}/6}\right)^2/2n < (2+\lambda)^7\log^6 n \times n^{1-2^{2.9}/6} = o(1) \ .$$

Thus,

$$\mathbf{P}\{Z_3 \ge 1 \mid Z_2\} \le \mathbf{E}\{Z_3 \mid Z_2\} = o(1) \ .$$

Thus,

$$\begin{aligned} \mathbf{P}\{Z_3 > 0\} \le\ & \mathbf{P}\left\{Z \ge n^{1-1/(6\times 2^{0.1})}\right\} \\ & + \mathbf{P}\left\{Z_1 \ge (2+\lambda)\log n \times n^{1-2^{0.9}/6} \mid Z \le n^{1-1/6\times 2^{0.1}}\right\} \\ & + \mathbf{P}\left\{Z_2 \ge (2+\lambda)^3\log^3 n \times n^{1-2^{1.9}/6} \mid Z_1 \le (2+\lambda)\log n \times n^{1-2^{0.9}/6}\right\} \\ & + \mathbf{P}\left\{Z_3 \ge 1 \mid Z_2 \le (2+\lambda)^3\log^3 n \times n^{1-2^{1.9}/6}\right\} \\ =\ & o(1). \qquad \square \end{aligned}$$

Thus far, we have shown that if $\alpha \le 0..306891\ldots$, the probability that the maximal displacement of any element is more than $\log_2\log n + C$ for a constant C depending upon α only tends to zero. This matches the lower bound that we will present later on. We will now fill the gap and show this result for all α.

Proof of Theorem 1.1. In the proof, we let $m = \lfloor \alpha n \rfloor$ without loss of generality. We define the level of an element as the number of probes required to locate it. The level is one if the element is stored at its original location. (Thus, the level is one more than the age of an element.) We define the level of a cell in the table as the level of the element occupying the cell if the cell is occupied, and zero otherwise. At time t, when the table holds t elements, we define

$$N_t(i) = \#\text{ elements of level} \geq i \ .$$

Note that $N_t(i)$ is monotone in t for fixed i. When inserting the tth element, let K_t be the number of cells probed. Clearly, K_t is geometric:

$$\mathbf{P}\{K_t = k\} = \left(1 - \frac{t-1}{n}\right)\left(\frac{t-1}{n}\right)^{k-1}, \quad k \geq 1 \ .$$

We begin with a rough tail bound for $N_t(i)$.

LEMMA 1.6. *Define*

$$\beta = \frac{2(1+\alpha)}{(1-\alpha)\log((1+\alpha)/2\alpha)} \ .$$

Then for all $t \leq m$, $i \geq 1$,

$$\mathbf{P}\left\{N_t(i) \geq \beta m \alpha^{i-1}\right\} \leq \mathbf{P}\left\{N_m(i) \geq \beta m \alpha^{i-1}\right\} \leq \exp\left(-\frac{1+\alpha}{1-\alpha} m \alpha^{i-1}\right) \ .$$

Proof. When we insert the tth element, we can increase the number of elements of level $\geq i$ by at most $(K_t - i)_+$. Therefore,

$$N_t(i) \leq \sum_{j=1}^{t} (K_j - i)_+ \ ,$$

where $K_1, K_2, \ldots, K_t$ are independent. As $K_1 \prec K_2 \prec \cdots \prec K_t$ (where $\prec$ denotes stochastic ordering), we see that

$$N_t(i) \prec \sum_{j=1}^{t} (K_{t,j} - i)_+ \ ,$$

where $K_{t,1}, \ldots, K_{t,t}$ are i.i.d. and distributed as K_t. We will use Chernoff bounding (Chernoff (1952); Hoeffding (1963); Azuma (1967); McDiarmid (1989, 1998)). Let $\lambda, u > 0$. Then

$$\begin{aligned}
\mathbf{P}\{N_t(i) \geq u\} &\leq \mathbf{P}\{N_m(i) \geq u\} \\
&\leq e^{-\lambda u}\left(\mathbf{E}\left\{e^{\lambda(K_m - i)_+}\right\}\right)^m \\
&\leq e^{-\lambda u}\left(\mathbf{P}\{K_m \leq i\} + \sum_{j=1}^{\infty} e^{\lambda j}\mathbf{P}\left\{K_m = i + j\right\}\right)^m \\
&\leq e^{-\lambda u}\left(1 + \sum_{j=1}^{\infty} e^{\lambda j}\left(1 - \frac{m-1}{n}\right)\left(\frac{m-1}{n}\right)^{i+j-1}\right)^m
\end{aligned}$$

$$
\begin{aligned}
&\le e^{-\lambda u}\left(1+\sum_{j=1}^{\infty} e^{\lambda j}\,(\alpha)^{i+j-1}\right)^m \\
&\le e^{-\lambda u}\left(1+\alpha^{i-1}\frac{e^{\lambda}\alpha}{1-e^{\lambda}\alpha}\right)^m \\
&= e^{-\lambda u}\left(1+\frac{1+\alpha}{1-\alpha}\alpha^{i-1}\right)^m \\
&\quad (\text{set } e^{\lambda}\alpha=(1+\alpha)/2) \\
&\le \exp\left(-u\log((1+\alpha)/2\alpha)+\frac{1+\alpha}{1-\alpha}\alpha^{i-1}m\right) \\
&= \exp\left(-\frac{1+\alpha}{1-\alpha}\alpha^{i-1}m\right) \\
&\quad \left(\text{set } u=\tfrac{2(1+\alpha)\alpha^{i-1}m}{(1-\alpha)\log((1+\alpha)/2\alpha)}\right) .
\end{aligned}
$$

This concludes the proof. □

Note that, for any given R, the cardinality of the R-head in the previous section is not more than $N_m(R)$. Assume that we were to start with an R-head of size $m' \le \alpha' n$, where R and α' are defined in Lemma 1.7. Then, by mimicking the argument of the previous section, we have the following.

LEMMA 1.7. *Define* $b=(2\lambda+2)\alpha'$ *and assume that* $b<1$. *Define*

$$D=\left\lfloor \log_2\left(\frac{2}{3\log(1/b)}\right)-0.1\right\rfloor$$

and

$$R=\lceil \lambda\log\left(\beta(2\lambda+3)\right)\rceil \ .$$

Let Z *be the number of elements in the* $(R+r)$*-head, with* $r=\lfloor \log_2\log n\rfloor + D$. *Then*

$$\lim_{n\to\infty}\mathbf{P}\left\{Z\ge\frac{nb^{2^r}}{2\lambda+2}\right\}=0 \ .$$

In particular,

$$\lim_{n\to\infty}\mathbf{P}\left\{Z\ge n^{1-1/(6\times 2^{0.1})}\right\}=0 \ .$$

Note in particular that the only difference between Lemma 1.4 and Lemma 1.7 is in the replacement of α in the definition of b by α'. The definition of λ is unaltered. Lemma 1.5 would then imply that with probability tending to one, $M_n \le R+r+2$, with r as in Lemma 1.7. Since the number of elements in the R-head is random, we use the following argument, based on Lemma 1.6. Define

$$\beta=\frac{2(1+\alpha)}{(1-\alpha)\log((1+\alpha)/2\alpha)} \ .$$

Then

$$
\begin{aligned}
&\mathbf{P}\ \{M_n > R+r+2\} \\
&\quad \le \mathbf{P}\left\{N_m(R)\ge\beta m\alpha^{R-1}\right\}+\mathbf{P}\{M_n>R+r+2\mid N_m(R)\le\beta m\alpha^{R-1}\} \\
&\quad \le \exp\left(-\frac{1+\alpha}{1-\alpha}m\alpha^{R-1}\right)+o(1) \ ,
\end{aligned}
$$

provided that $\beta m\alpha^{R-1} \le \alpha' n$, where $\alpha' = 1/(2\lambda+3)$ (to make b in Lemma 1.7 less than one). But $\beta m\alpha^{R-1} \le \beta n\alpha^R$, and thus it suffices to set

$$R = \left\lceil \frac{\log\left(\beta(2\lambda+3)\right)}{\log(1/\alpha)} \right\rceil = \lceil \lambda \log\left(\beta(2\lambda+3)\right) \rceil \ .$$

With this choice of R, and the choice of r given in Lemma 1.7, we thus conclude that

$$\lim_{n\to\infty} \mathbf{P}\{M_n > R + r + 2\} = 0 \ .$$

This concludes the proof of Theorem 1.1. □

Proof of Theorem 1.2. We prove the theorem by construction. This is done by identifying a subset of elements that must be of age at least 1, a further subset of age 2, and so forth. We show that this process can be carried out at least k times with high probability until we run out of elements, where k is of the order of $\log\log n$. We consider all values $X_{i,0}$, $1 \le i \le m$, first. Consider that ball i is dropped in urn $X_{i,0}$. If an urn receives j elements, then at least $j-1$ of them must move on, and will have an age at least equal to 1. Who moves on depends upon the tiebreaking strategy, but in our analysis, it only matters to know how many move on. We introduce A_r, the number of elements that are marked in the rth step. A_1 is the number of elements of age at least 1 in the process above. We formally set $A_0 = m$. Given A_{r-1}, we take the A_{r-1} elements of age at least $r-1$ (note: these are not the only ones of age at least $r-1$) and look at their $X_{i,r}$ values, with the number of i's clearly being A_{r-1}. We consider the subset that has to move on, so only urns with at least two elements can be of any use. Note that in view of the tiebreaking policy, an earlier element may move on. But in any case, if an urn receives j elements from the A_{r-1}, at least $j-1$ of them must move on and increase their age by one. These $j-1$ elements are collected and form a further subset of size A_r, consisting entirely of elements of age at least r.

We return now to our process A_r. We observe that A_r has the (A_{r-1}, n) urn distribution. The inequalities (1) suggest natural bounds for A_r. We define an integer sequence a_r such that with high probability, $a_r \le A_r$. We have $a_0 = m = \lceil \alpha n \rceil$. Then set

$$a_{r+1} = a_r^2/8n \ .$$

Note that

$$a_r = 8n(a_0/8n)^{2^r} \ge 8n(\alpha/8)^{2^r} \ .$$

Define the events

$$E_r = \cap_{j\le r}[a_j \le A_j]$$

and let $(.)^c$ denote the complement of an event. Observe the following:

$$\mathbf{P}\{E_r^c\} \le \mathbf{P}\{E_1^c\} + \sum_{j=2}^{r} \mathbf{P}\{E_j^c \mid E_0, \ldots, E_{j-1}\} = \sum_{j=2}^{r} \mathbf{P}\{E_j^c \mid E_{j-1}\} \ .$$

Also, if r is so small that at all times $a_{r-1} \ge 4$ (a condition that is needed so that we may apply the inequalities derived in section 1.1), we have

$$\begin{aligned}\mathbf{P}\{E_r^c \mid E_{r-1}\} &\le \mathbf{P}\left\{[A_r < a_r] \mid a_{r-1} \le A_{r-1}\right\} \\ &\le \mathbf{P}\left\{A_r < \frac{1}{2}\mathbf{E}\{A_r \mid a_{r-1} \le A_{r-1}\} \mid a_{r-1} \le A_{r-1}\right\} \ ,\end{aligned}$$

provided that $a_r \le (1/2)\mathbf{E}\{A_r \mid a_{r-1} \le A_{r-1}\}$. But this follows from $a_r = a_{r-1}^2/8n = (1/2)a_{r-1}^2/4n \le \mathbf{E}\{A_r \mid a_{r-1} \le A_{r-1}\}$. We let A have the $(\lceil a_{r-1} \rceil, n)$ urn distribution. Thus,

$$\begin{aligned}
\mathbf{P}\{E_r^c \mid E_{r-1}\} &\le \mathbf{P}\left\{A < \frac{1}{2}\mathbf{E}\{A\}\right\} \\
&\le \frac{\lceil a_{r-1} \rceil}{2(\mathbf{E}\{A\}/2)^2} \\
&\le \frac{2\lceil a_{r-1} \rceil}{((\lceil a_{r-1} \rceil)^2/4n)^2} \\
&\le \frac{32n^2}{a_{r-1}^3} \\
&\le \frac{(8/\alpha)^{3\times 2^{r-1}}}{16n} \ .
\end{aligned}$$

Therefore,

$$\mathbf{P}\{E_r^c\} \le \sum_{j=0}^{r-1} \frac{(8/\alpha)^{3\times 2^j}}{16n} = \frac{1}{16n}\sum_{j=0}^{r-1}(8/\alpha)^{3\times 2^j} \le \frac{(8/\alpha)^{3\times 2^{r-1}}}{16n(1-(\alpha/8)^3)} \ .$$

Set $r = \lceil \log_2(c \log n) \rceil$ for $c > 0$, and note that the upper bound is not more than

$$\frac{n^{3c\log(8/\alpha)-1}}{16(1-\alpha/8)} \ ;$$

this tends to zero if $c < 1/3\log(8/\alpha)$. With that choice of r, we note that

$$a_r \ge \frac{8n}{(8/\alpha)^{3\times 2^r}} \ge \frac{8n}{n^{6c\log(8/\alpha)}} \ge 8 \ ,$$

provided we take $c = 1/6\log(8/\alpha)$. With such a choice, we then have

$$\mathbf{P}\{A_r = 0\} \le \mathbf{P}\{A_r < a_r\} \le \mathbf{P}\{E_r^c\} \le \frac{1}{16(1-(\alpha/8)^3)\sqrt{n}} \ . \qquad \square$$

REFERENCES

O. Amble and D. E. Knuth (1974), *Ordered hash tables*, Comput. J., 17, pp. 135–142.

Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal (1999), *Balanced allocations*, SIAM J. Comput., 29, pp. 180–200.

K. Azuma (1967), *Weighted sums of certain dependent random variables*, Tohoku Math. J., 37, pp. 357–367.

A. D. Barbour, L. Holst, and S. Janson (1992), *Poisson Approximation*, Oxford University Press, Oxford.

A. Z. Broder and A. R. Karlin (1990), *Multilevel adaptive hashing*, in Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, pp. 43–53.

A. Brodnik and J. I. Munro (1999), *Membership in constant time and almost-minimum space*, SIAM J. Comput., 28, pp. 1627–1640.

P. Celis, P.-Å. Larson, and J. I. Munro (1985), *Robin Hood hashing*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, pp. 281–288.

P. Celis (1986), *Robin Hood Hashing*, Technical Report CS-86-14, Computer Science Department, University of Waterloo, Waterloo, ON, Canada.

H. Chernoff (1952), *A measure of asymptotic efficiency of tests of a hypothesis based on the sum of observations*, Ann. Math. Statist., 23, pp. 493–507.

A. Czumaj and V. Stemann (1997), *Randomized allocation processes*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, IEEE Computer Society Press, Los Alamitos, CA, pp. 194–203.

L. Devroye (1985), *The expected length of the longest probe sequence when the distribution is not uniform*, J. Algorithms, 6, pp. 1–9.

M. Dietzfelbinger and F. Meyer auf de Heide (1990), *A new universal class of hash functions and dynamic hashing in real time*, in Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP '90), Lecture Notes in Comput. Sci. 443, Springer-Verlag, New York, pp. 6–19.

M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger (1992), *Polynomial hash functions are reliable (extended abstract)*, in Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP '92), Lecture Notes in Comput. Sci. 623, Springer-Verlag, New York, pp. 235–246.

M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf de Heide, H. Rohnert, and R. E. Tarjan (1994), *Dynamic perfect hashing: Upper and lower bounds*, SIAM J. Comput., 23, pp. 738–761.

B. Efron and C. Stein (1981), *The jackknife estimate of variance*, Ann. Statist., 9, pp. 586–596.

P. Flajolet, P. V. Poblete, and A. Viola (1998), *On the analysis of linear probing hashing*, Algorithmica, 22, pp. 490–515.

M. L. Fredman, J. Komlós, and E. Szemerédi (1984), *Storing a sparse table with $O(1)$ worst case access time*, J. ACM, 31, pp. 538–544.

G. H. Gonnet (1981), *Expected length of the longest probe sequence in hash code searching*, J. ACM, 28, pp. 289–304.

G. H. Gonnet and R. Baeza-Yates (1991), *Handbook of Algorithms and Data Structures*, 2nd ed., Addison-Wesley, Reading, MA.

G. R. Grimmett and D. R. Stirzaker (1992), *Probability and Random Processes*, Oxford University Press, Oxford.

W. Hoeffding (1963), *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58, pp. 13–30.

D. E. Knuth (1998), *The Art of Computer Programming, Vol.* 3: *Sorting and Searching*, 2nd ed., Addison-Wesley, Reading, MA.

C. McDiarmid (1989), *On the method of bounded differences*, in Surveys in Combinatorics, J. Siemons, ed., London Math. Soc. Lecture Note Ser. 141, Cambridge University Press, Cambridge, UK, pp. 148–188.

C. McDiarmid (1998), *Concentration*, in Probabilistic Methods for Algorithmic Discrete Mathematics, M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, eds., Springer-Verlag, New York, pp. 195–248.

M. Mitzenmacher (1997), *Studying Balanced Allocations with Differential Equations*, Technical Note 1997024, Digital Equipment Corporation Systems Research Center, Palo Alto, CA.

M. Mitzenmacher, A. W. Richa, and R. Sitaraman (2000), *The Power of Two Random Choices: A Survey of Techniques and Results*, Technical Report.

R. Pagh and F. F. Rodler (2001), *Cuckoo Hashing*, BRICS Report Series RS-01-32, Department of Computer Science, University of Aarhus, Aarhus, Denmark.

P. V. Poblete and J. I. Munro (1989), *Last-come-first-served hashing*, J. Algorithms, 10, pp. 228–248.

J. M. Steele (1986), *An Efron-Stein inequality for nonsymmetric statistics*, Ann. Statist., 14, pp. 753–758.

A. Viola and P. V. Poblete (1998), *Analysis of linear probing hashing with buckets*, Algorithmica, 21, pp. 37–71.

J. S. Vitter and P. Flajolet (1990), *Average-case analysis of algorithms and data structures*, in Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, ed. J. van Leeuwen, ed., MIT Press, Amsterdam, pp. 431–524.

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 937–951

ONLINE ROUTING IN TRIANGULATIONS*

PROSENJIT BOSE† AND PAT MORIN†

Abstract. We consider online routing algorithms for routing between the vertices of embedded planar straight line graphs. Our results include (1) two deterministic memoryless routing algorithms, one that works for all Delaunay triangulations and the other that works for all regular triangulations; (2) a randomized memoryless algorithm that works for all triangulations; (3) an $O(1)$ memory algorithm that works for all convex subdivisions; (4) an $O(1)$ memory algorithm that approximates the shortest path in Delaunay triangulations; and (5) theoretical and experimental results on the competitiveness of these algorithms.

Key words. routing, online algorithms, Delaunay triangulations, shortest path, spanning path

AMS subject classifications. 65D18, 90B18

DOI. 10.1137/S0097539700369387

1. Introduction.

Path finding, or routing, is central to a number of fields, including geographic information systems, urban planning, robotics, and communication networks. In many cases, knowledge about the environment in which routing takes place is not available beforehand, and the vehicle/robot/packet must learn this information through exploration. Algorithms for routing in these types of environments are referred to as *online* [3] routing algorithms.

In this paper we consider online routing in the following abstract setting: The environment is a planar straight line graph [17], T, with n vertices, whose edges are weighted by the Euclidean distance between their endpoints, the source v_{src} and destination v_{dst} are vertices of T, and a packet can only move on edges of T. Initially, a packet only knows v_{src}, v_{dst}, and $N(v_{\text{src}})$, where $N(v)$ denotes the set of vertices adjacent to v.

We classify online routing algorithms based on their use of memory and/or randomization. Define v_{cur} as the vertex at which the packet is currently stored. A routing algorithm is called *memoryless* if the next step taken by a packet depends only on v_{cur}, v_{dst}, and $N(v_{\text{cur}})$. An algorithm is *randomized* if the next step taken by a packet is chosen randomly from $N(v_{\text{cur}})$. A randomized algorithm is memoryless if the distribution used to choose from $N(v_{\text{cur}})$ is a function only of v_{cur}, v_{dst}, and $N(v_{\text{cur}})$.

The justification for studying the memory requirements of routing algorithms comes from communication networks, in which memory used by an algorithm results in header information that travels with a packet. Since this information is used only for routing purposes and is of no use to the sender or receiver, it effectively produces a decrease in communication bandwidth.

For an algorithm $\mathcal{A}$ we say that a graph *defeats* $\mathcal{A}$ if there is a source/destination pair such that a packet never reaches the destination when beginning at the source. If $\mathcal{A}$ finds a path P from v_{src} to v_{dst}, we call P the $\mathcal{A}$ path from v_{src} to v_{dst}. Here

*Received by the editors March 15, 2000; accepted for publication (in revised form) January 28, 2004; published electronically May 25, 2004. This research was supported by the Natural Sciences and Engineering Research Council of Canada.
http://www.siam.org/journals/sicomp/33-4/36938.html
†School of Computer Science, Carleton University, 1125 Colonel By Dr., Ottawa, Canada, K1S 5B6 (jit@scs.carleton.ca, morin@cs.carleton.ca).

we use the term *path* in an intuitive sense rather than a strict graph theoretic sense, since P may visit the same vertex more than once.

In this paper we also consider, as a special case, a class of "well-behaved" triangulations. The *Voronoi diagram* [16] of S is a partitioning of space into cells such that all points within a Voronoi cell are closer to the same element $p \in S$ than any other point in S. The *Delaunay triangulation* is the straight line face dual of the Voronoi diagram; i.e., two points in S have an edge between them in the Delaunay triangulation if their Voronoi cells have an edge in common.

In this paper we consider several different routing algorithms and compare their performance empirically. In particular, we describe

1. a memoryless algorithm that is not defeated by any Delaunay triangulation;
2. a memoryless algorithm that is not defeated by any regular triangulation;
3. a memoryless randomized algorithm that uses 1 random bit per step and is not defeated by any triangulation;
4. an algorithm that only remembers a constant number of vertex locations that is not defeated by any convex subdivision (we say that such an algorithm *uses* $O(1)$ *memory*);
5. an algorithm for Delaunay triangulations that uses $O(1)$ memory in which a packet never travels more than a constant times the Euclidean distance between v_{src} and v_{dst}; and
6. a theoretical and empirical study of the quality (length) of the paths found by these algorithms.

The first four routing algorithms are described in section 2. Section 3 presents theoretical and empirical results on the length of the paths found by these algorithms and describes our algorithm for Delaunay triangulations. A discussion of related work is provided in section 4. Finally, section 5 summarizes our results and describes directions for future research.

2. Four simple algorithms. In this section we describe four online routing algorithms and prove theorems about which types of graphs never defeat them. We begin with the simplest (memoryless) algorithms and proceed to the more complex algorithms.

However, before beginning we should note that deterministic memoryless algorithms have some inherent limitations. Consider what happens when such an algorithm tries to route from one of the vertices of the outer face to v_{dst} in the graphs shown in Figure 2.1. In each of these graphs, the neighborhoods of the corner vertices look the same. Therefore, any deterministic memoryless algorithm must make the same decisions at the corners in each of the graphs. There are then four cases to consider.

1. At all three corners, the algorithm chooses to use an edge of the convex hull. In this case, the algorithm will fail on the graph in Figure 2.1.a since it will never enter the interior of the convex hull and will therefore never reach v_{dst}.
2. At two of the corners, the algorithm chooses to use an edge of the convex hull and at the third corner it does not. We can assume, without loss of generality that the third corner is the bottom right corner. In this case, the algorithm will fail on the graph shown in Figure 2.1.b since the only way to reach v_{dst} from the convex hull is via one of the two paths in the other two corners.
3. At one of the corners, the algorithm chooses to use an edge of the convex hull and at the other two corners it does not. We may assume without loss of generality that the corner that uses the interior edge is the top corner. In

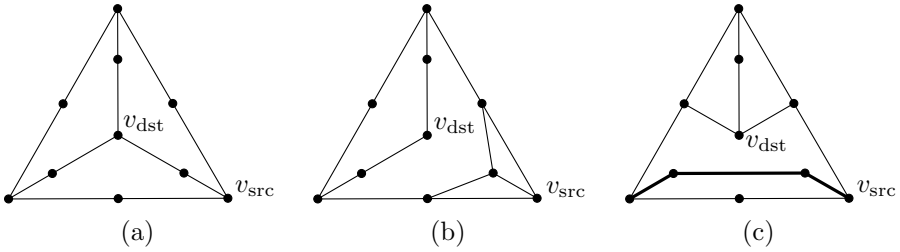


FIG. 2.1. *No deterministic memoryless routing algorithm can work for all* 2*-connected graphs.*

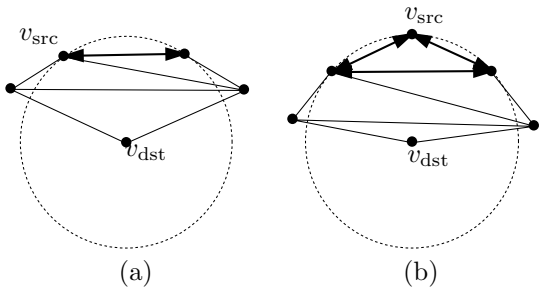


FIG. 2.2. *Triangulations that defeat the greedy routing algorithm.*

this case, the algorithm will fail on the graph in Figure 2.1.c since it will get trapped cycling among the edges shown in bold.

4. At all of the corners, the algorithm chooses not to use an edge of the convex hull. In this case the algorithm will also fail on the graph in Figure 2.1.c for the same reasons as in case 3.

Since the graphs in Figure 2.1 are all 2-connected we have the following negative result.

LEMMA 2.1. *No deterministic memoryless algorithm works for all* 2*-connected planar graphs.*

2.1. Greedy routing. The *greedy routing* algorithm always moves the packet to the neighbor $gdy(v_{\text{cur}})$ of v_{cur} that minimizes $dist(gdy(v_{\text{cur}}), v_{\text{dst}})$, where $dist(p, q)$ denotes the Euclidean distance between p and q. In the case of ties, one of the vertices is chosen arbitrarily. The greedy routing algorithm can be defeated by a triangulation T in two ways (the first way is an important special case of the second): (1) the packet can get trapped moving back and forth on an edge of the triangulation (Figure 2.2.a), or (2) the packet can get trapped on a cycle of three or more vertices (Figure 2.2.b). However, as the following theorem shows, neither of these situations can occur if T is a Delaunay triangulation.

THEOREM 2.2. *There is no point set whose Delaunay triangulation defeats the greedy routing algorithm.*

Proof. We proceed by showing that every vertex v of T has a neighbor that is strictly closer to v_{dst} than v is. Thus, at each routing step, the packet gets closer to v_{dst} and therefore, after at most n steps, reaches v_{dst}. Refer to Figure 2.3.

Consider the Voronoi diagram [16] $VD(T)$ of the vertices of T and let e be the first edge of $VD(T)$ intersected by the directed line segment (v, v_{dst}). Note that e is on the boundary of two Voronoi cells, one for v and one for some other vertex u, and the

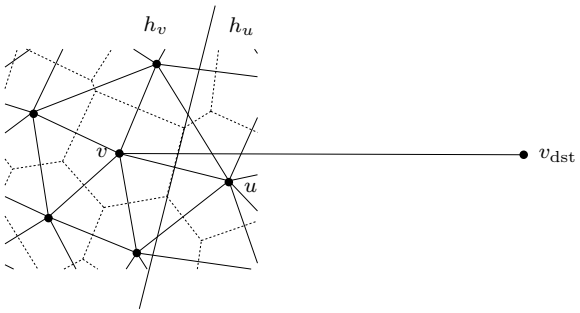


FIG. 2.3. *The proof of Theorem* 2.2.

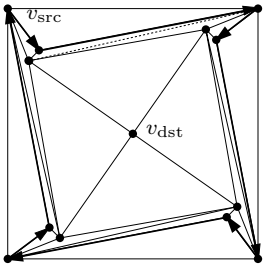


FIG. 2.4. *A triangulation that defeats the compass routing algorithm.*

supporting line of e partitions the plane into two open half planes $h_v = \{p : dist(p, v) < dist(p, u)\}$ and $h_u = \{p : dist(p, u) < dist(p, v)\}$. Since the Voronoi diagram is the straight line face dual of the Delaunay triangulation, the edge $(u, v) \in T$. Also, by the choice of e, $v_{\text{dst}} \in h_u$, i.e., $dist(u, v_{\text{dst}}) < dist(v, v_{\text{dst}})$. □

2.2. Compass routing. The *compass routing* algorithm always moves the packet to the vertex $cmp(v_{\text{cur}})$ that minimizes the angle $\angle v_{\text{dst}}, v_{\text{cur}}, cmp(v_{\text{cur}})$ over all vertices adjacent to v_{cur}. Here the angle is taken to be the smaller of the two angles as measured in the clockwise and counterclockwise directions. In the case of ties, one of the (at most 2) vertices is chosen using some arbitrary deterministic rule.

One might initially believe (as we did) that compass routing can always be used to find a path between any two vertices in a triangulation. However, the triangulation in Figure 2.4 defeats compass routing. When starting from one of the vertices on the outer face of T, and routing to v_{dst}, the compass routing algorithm gets trapped on the cycle shown in bold. The following lemma shows that any triangulation that defeats compass routing causes the packet to get trapped in a cycle.

LEMMA 2.3. *Let T be a triangulation that defeats compass routing, and let v_{dst} be a vertex such that compass routing fails to route a packet to v_{dst} when given some other vertex as the source. Then there exists a cycle $C = v_0, \ldots, v_{k-1}$ ($k \geq 3$) in T such that $cmp(v_i) = v_{i+1}$ for all $0 \leq i < k$.*[1]

Proof. Since T defeats compass routing, and the compass routing algorithm makes the same decision each time it visits a vertex, either there is an edge (u, v) such that $cmp(u) = v$ and $cmp(v) = u$, or there is the situation described in the lemma. We prove that there can be no such edge (u, v). Suppose such an edge (u, v) does exist.

[1]Here and henceforth, all subscripts are assumed to be taken mod$k$.

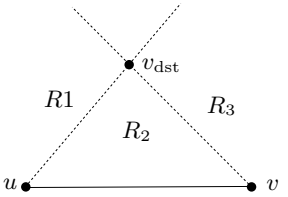


FIG. 2.5. *The proof of Lemma* 2.3.

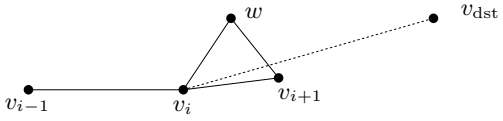


FIG. 2.6. *The proof of Lemma* 2.4.

Then there is a triangle (u, v, w) in T such that w is in the same half plane bounded by the line through u and v as v_{dst}. Referring to Figure 2.5, the vertex w must be in one of the regions 1, 2, or 3. But this is a contradiction, since if w is in region 1, then $cmp(v) = w$; if w is in region 2, then $cmp(u) = w$ (and $cmp(v) = w$); and if w is in region 3, then $cmp(u) = w$. □

We call such a cycle, C, a *trapping cycle* in T for v_{dst}. Next we characterize trapping cycles in terms of a visibility property of triangulations. Let t_1 and t_2 be two triangles in T. Then we say that t_1 *obscures* t_2 with respect to viewpoint v_{dst} if there exists a ray originating at v_{dst} that strikes t_1 first and then t_2. Let u and v be any two vertices of T such that $cmp(u) = v$. Then define $\triangle uv$ as the triangle of T that is contained in the closed half plane bounded by the line through uv, that contains the edge uv, and that contains v_{dst}. We obtain the following useful characterization of trapping cycles.

LEMMA 2.4. *Let T be a triangulation that defeats compass routing and let $C = v_0, \ldots, v_{k-1}$ be a trapping cycle in T for vertex v_{dst}. Then $\triangle v_i v_{i+1}$ is either identical to or obscures $\triangle v_{i-1} v_i$ for all $0 \leq i < k$.*

Proof. Refer to Figure 2.6. Assume that $\triangle v_i v_{i+1}$ and $\triangle v_{i-1} v_i$ are not identical; otherwise the lemma is trivially true. Let w be the third vertex of $\triangle v_i v_{i+1}$. Then w cannot lie in the cone defined by v_{dst}, v_i, and v_{i+1}; otherwise we would have $cmp(v_i) = w$. But then the line segment joining w and v_{i+1} obscures v_i and hence $\triangle v_i v_{i+1}$ obscures $\triangle v_{i-1} v_i$. □

A *regular triangulation* [18] is a triangulation obtained by orthogonal projection of the faces of the lower hull of a three-dimensional polytope onto the plane. Note that the Delaunay triangulation is a special case of a regular triangulation in which the vertices of the polytope all lie on a paraboloid. Edelsbrunner [8] showed that if T is a regular triangulation, then T has no set of triangles that obscure each other cyclically from *any* viewpoint. This result, combined with Lemma 2.4, yields our main result on compass routing.

THEOREM 2.5. *There is no regular triangulation that defeats the compass routing algorithm.*

2.3. Randomized compass routing. In this section, we consider a randomized routing algorithm that is not defeated by any triangulation. Let $cw(v)$ be the vertex in

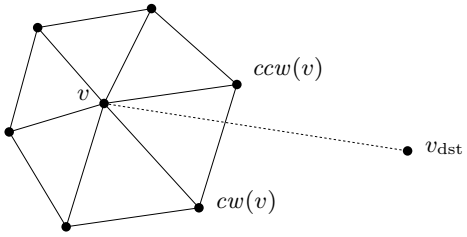


FIG. 2.7. *Definition of $cw(v)$ and $ccw(v)$.*

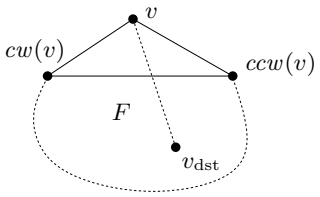


FIG. 2.8. *The proof of Theorem* 2.6.

$N(v)$ that minimizes the clockwise angle $\angle v_{\text{dst}}, v, cw(v)$ and let $ccw(v)$ be the vertex in $N(v)$ that minimizes the counterclockwise angle $\angle v_{\text{dst}}, v, ccw(v)$ (see Figure 2.7). Then the *randomized compass routing* (RCR) algorithm moves the packet to one of $\{cw(v_{\text{cur}}), ccw(v_{\text{cur}})\}$ with equal probability.

Before we can make statements about which triangulations defeat randomized compass routing, we must define what it means for a triangulation to defeat a randomized algorithm. We say that a triangulation T defeats a (randomized) routing algorithm if there exists a pair of vertices v_{src} and v_{dst} of T such that a packet originating at v_{src} with destination v_{dst} has probability 0 of reaching v_{dst} in any finite number of steps. Note that proving that a triangulation T does not defeat a memoryless routing algorithm implies that a packet reaches its destination with probability 1.

It is well known that a random walk will eventually visit every vertex of a connected graph. Thus, a random walk is a randomized routing algorithm that is not defeated by any connected graph. However, this result is not satisfactory for two reasons: (1) Because a random walk does not take the destination into account, the path it takes is by no means direct, and (2) the number of random bits required at each step of a random walk is $\log d_{\text{cur}}$, where d_{cur} is the degree of the current vertex. In contrast, randomized compass routing requires only 1 bit at each step and is more likely to take a direct path to the destination vertex.

The following theorem shows the versatility of randomized compass routing.

THEOREM 2.6. *There is no triangulation that defeats the randomized compass routing algorithm.*

Proof. Assume, by way of contradiction that a triangulation T exists that defeats the randomized compass routing algorithm. Then there is a vertex v_{dst} of T and a minimal set S of vertices such that (1) $v_{\text{dst}} \notin S$, (2) the subgraph H of T induced by S is connected, and (3) for every $v \in S$, $cw(v) \in S$ and $ccw(v) \in S$.

Refer to Figure 2.8 for what follows. The vertex v_{dst} lies in some face F of H. Let v be a vertex on the boundary of F such that the line segment (v, v_{dst}) is contained in F. Such a vertex is guaranteed to exist [5]. The two neighbors of v on the boundary of

F must be $cw(v)$ and $ccw(v)$, and these cannot be the same vertex (since F contains (v, v_{dst}) in its interior). Note that, by the definitions of $cw(v)$ and $ccw(v)$, and by the fact that T is a triangulation, the triangle $(cw(v), v, ccw(v))$ is in T. But this is a contradiction, since then v is not on the boundary of F. □

2.4. Right-hand routing. The folklore "right-hand rule" for exploring a maze states that if a player in a maze walks around never lifting her right-hand from the wall, then she will eventually visit every wall in the maze. More specifically, if the maze is the face of a connected planar straight line graph, the player will visit every edge and vertex of the face [2].

Let T be any convex subdivision. Consider the planar subdivision T' obtained by deleting from T all edges that properly intersect the line segment joining v_{src} and v_{dst}. Because of convexity, T' is connected, and v_{src} and v_{dst} are on the boundary of the same face F of T'. The *right-hand routing* algorithm uses the right-hand rule on the face F to route from v_{src} to v_{dst}. Right-hand routing is easily implemented using only $O(1)$ additional memory by remembering v_{src}, v_{dst}, and the last vertex visited.

THEOREM 2.7. *There is no convex subdivision that defeats the right-hand routing algorithm.*

3. Competitiveness of paths. Thus far we have considered only the question of whether routing algorithms can find *a* path between any two vertices in T. An obvious direction for research is to consider the length of the path found by a routing algorithm. We say that a routing algorithm $\mathcal{A}$ is c-competitive for T if for any pair $(v_{\text{src}}, v_{\text{dst}})$ in T, the length (sum of the edge lengths) of the path between v_{src} and v_{dst} found by $\mathcal{A}$ is at most c times the length of the shortest path between v_{src} and v_{dst} in T. In the case of randomized algorithms, we use the expected length of the path. We say that $\mathcal{A}$ has a competitive ratio of c if it is c-competitive.

This section addresses questions about the competitive ratio of the algorithms described so far, as well as a new algorithm specifically targeted for Delaunay triangulations. We present theoretical as well as experimental results.

3.1. Negative results. It is not difficult to contrive triangulations for which none of our algorithms is c-competitive for any constant c. Thus it is natural to restrict our attention to a well-behaved class of triangulations. Unfortunately, even for Delaunay triangulations none of the algorithms described so far is c-competitive.

THEOREM 3.1. *There exist Delaunay triangulations for which none of the greedy, compass, randomized compass, or right-hand routing algorithms is c-competitive for any constant c.*

Proof. We begin with greedy routing. Consider the set of points that are placed on a circle and then triangulated to obtain the zig-zag triangulation T shown in Figure 3.1.a. Since the points are cocircular, this is a valid Delaunay triangulation. The points are placed so that each vertex v has a neighbor on the opposite side of the line through v_{src} and v_{dst} that is closer to v_{dst} than v's two neighbors on the same side of the line.

Note that there exists a path between v_{src} and v_{dst} of length approximately $(\pi/2) \cdot dist(v_{\text{src}}, v_{\text{dst}})$, and this is therefore an upper bound on the length of the shortest path between v_{src} and v_{dst}. The length of the "zig-zag" path that uses the diagonals of T between v_{src} and v_{dst} is $\Theta(n) \cdot dist(v_{\text{src}}, v_{\text{dst}})$, and this is the path taken by the greedy routing algorithm. Thus, greedy routing is not c-competitive for this triangulation.

To show that compass routing is not c-competitive, we again consider a set of cocircular points and make a zig-zag triangulation. Let v_{cur} be any point on the

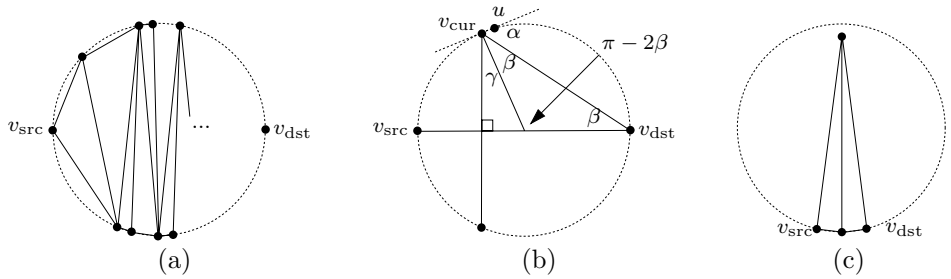


FIG. 3.1. *The proof of Theorem* 3.1.

circle with diameter $v_{\text{src}}, v_{\text{dst}}$. Consider the angle α between the tangent line passing through v_{cur} and the line through v_{src} and v_{dst}. Compare this with the angle between the line perpendicular to v_{src} and v_{dst} that passes through v_{cur} and the line through v_{src} and v_{dst}. Referring to Figure 3.1.b, we have

$$\alpha = \pi/2 - \beta, \tag{3.1}$$

$$\gamma = \pi/2 - 2\beta, \tag{3.2}$$

and therefore $\gamma + \beta = \pi/2 - \beta = \alpha$, i.e., the two angles are equal. Thus if compass routing were to choose between the tangent line and the line crossing the circle, it would be a tie. Now, by placing a point u on the circle close to v_{cur} we can make $\angle u, v_{\text{cur}}, v_{\text{dst}} = \alpha - \epsilon$ for arbitrarily small $\epsilon > 0$. Similarly, by placing a point v_{nxt} on the opposite side of the circle we can make $\angle v_{\text{nxt}}, v_{\text{cur}}, v_{\text{dst}} = \alpha - \epsilon - \delta$ for arbitrarily small $\delta > 0$, so that $cmp(v_{\text{cur}}) = v_{\text{nxt}}$. Since ϵ and δ can be arbitrarily small, we can repeat this construction as often as we like, thereby making the compass routing path arbitrarily long.

To see that randomized compass routing and right-hand routing are not c-competitive, consider a configuration of points like that in Figure 3.1.c. By making v_{src} and v_{dst} almost collinear with a third point, it is possible to produce arbitrarily long thin triangles that make the length of the path found by right-hand routing arbitrarily long. Furthermore, in this configuration the probability that the randomized compass routing path is the same as the right-hand path is 1/2, and thus the expected length of the randomized compass path can be arbitrarily large. □

3.2. A *c*-competitive algorithm for Delaunay triangulations. Since none of the algorithms described in section 2 is competitive, even for Delaunay triangulations, an obvious question is whether there exists any algorithm that is competitive for Delaunay triangulations. In this section we answer this question in the affirmative. In fact, we prove an even stronger result by giving an algorithm that finds a path whose cost is at most a constant times $dist(v_{\text{src}}, v_{\text{dst}})$.

Our algorithm is based on the remarkable proof of Dobkin, Friedman, and Supowit [7] that the Delaunay triangulation approximates the complete Euclidean graph to within a constant factor in terms of shortest path length. In the following we will use the notation $\text{x}(p)$ (resp., $\text{y}(p)$) to denote the x-coordinate (resp., y-coordinate) of the point p and the notation $|X|$ to denote the Euclidean length of the path X.

Consider the directed line segment from v_{src} to v_{dst}. This segment intersects regions of the Voronoi diagram in some order, say $R_0, \ldots, R_{m-1}$, where R_0 is the Voronoi region of v_{src} and R_{m-1} is the Voronoi region of v_{dst}. The *Voronoi routing* algorithm for Delaunay triangulations moves the packet from v_{src} to v_{dst} along the

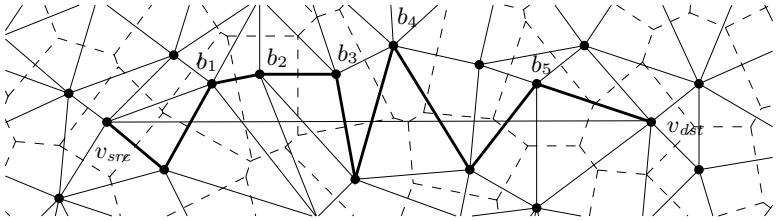


FIG. 3.2. *A path obtained by the Voronoi routing algorithm.*

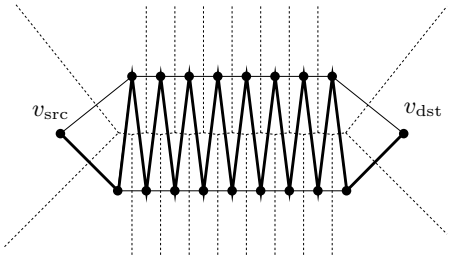


FIG. 3.3. *The Voronoi routing algorithm is not c-competitive for all Delaunay triangulations.*

path $v_0, \ldots, v_{m-1}$, where v_i is the site defining R_i. An example of a path obtained by the Voronoi routing algorithm is shown in Figure 3.2. Since the Voronoi region of a vertex v can be computed given only the neighbors of v in the Delaunay triangulation, it follows that the Voronoi routing algorithm is an $O(1)$ memory routing algorithm.

The Voronoi routing algorithm on its own is not c-competitive for all Delaunay triangulations, as can be seen from Figure 3.3.

However, it does have some properties that allow us to derive a c-competitive algorithm. As with right-hand routing, let T' be the graph obtained from T by removing all edges of T that properly intersect the segment $(v_{\text{src}}, v_{\text{dst}})$, and let F be the face of T' that contains both v_{src} and v_{dst}. Assume without loss of generality that v_{src} and v_{dst} both lie on the x-axis and that $\text{x}(v_{\text{src}}) < \text{x}(v_{\text{dst}})$. The following two lemmas follow from the work of Dobkin, Friedman, and Supowit [7].

LEMMA 3.2. *The Voronoi path is x-monotone, i.e.,* $\text{x}(v_i) < \text{x}(v_j)$ *for all* $i < j$.

LEMMA 3.3. *Let* P' *be the collection of maximal subpaths of* $v_0, \ldots, v_{m-1}$ *that remain above the x-axis, i.e.,* $P' = \{v_i, \ldots, v_j : \text{y}(v_{i-1}) < 0$ *and* $\text{y}(v_{j+1}) < 0$ *and* $\text{y}(v_k) \geq 0$ *for all* $i \leq k \leq j\}$. *Then* $\sum_{X \in P'} |X| \leq (\pi/2) \cdot dist(v_{\text{src}}, v_{\text{dst}})$.

Let $b_0, \ldots, b_{l-1}$ be the subsequence of vertices of $v_0, \ldots, v_{m-1}$ that are above or on the segment $(v_{\text{src}}, v_{\text{dst}})$. (Refer to Figure 3.2.) Consider two vertices $b_i = v_j$ and $b_{i+1} = v_k$, where $k \neq j+1$; i.e., the Voronoi path between b_i and b_{i+1} is not a direct edge. Let $P_V = (b_i = p_0, \ldots, p_x = b_{i+1})$ be the portion of the Voronoi path between b_i and b_{i+1} and let $P_F = (b_i = q_0, \ldots, q_y = b_{i+1})$ be the upper boundary of F between b_i and b_{i+1} (see Figure 3.4). Then the following holds.

LEMMA 3.4. *Let* $c_{\text{dfs}} = (1 + \sqrt{5})\frac{\pi}{2}$.[2] *Then* $|P_V| \leq c_{\text{dfs}} \cdot (\text{x}(b_i) - \text{x}(b_i))$ *or* $|P_F| \leq c_{\text{dfs}} \cdot (\text{x}(b_i) - \text{x}(b_i))$.

Proof. Let $c_0, \ldots, c_z$ be the lower convex hull of P_F, and let P_j be the Voronoi

[2]We call c_{dfs} the Dobkin–Friedman–Supowit constant [7].

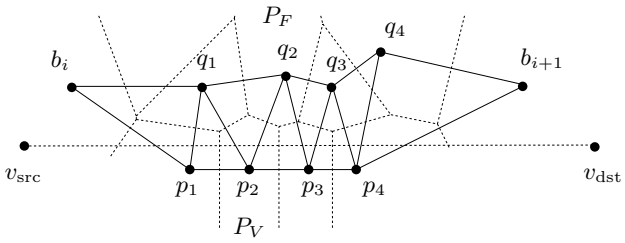


FIG. 3.4. *Definitions of P_V and P_F.*

path from c_j to c_{j+1}. Dobkin et al. prove that

$$|P_V| \leq c_{\text{dfs}} \cdot (\text{x}(b_{i+1} - \text{x}(b_i)) \quad \text{or} \quad \sum_{j=0}^{z-1} |P_j| \leq c_{\text{dfs}} \cdot (\text{x}(b_{i+1} - \text{x}(b_i)). \tag{3.3}$$

We claim that this implies Lemma 3.4 and prove this by showing that P_j visits all vertices of P_F between c_j and c_{j+1}. Thus, by the triangle inequality,

$$|P_F| \leq \sum_{j=0}^{z-1} |P_j|. \tag{3.4}$$

Refer to Figure 3.5 for what follows. Assume for the sake of contradiction that there is a vertex q in P_F between c_j and c_{j+1} that is not in P_j. As part of their proof, Dobkin et al. show that P_j remains entirely above the segment (c_j, c_{j+1}). Therefore, let Q be the polygon bounded by P_j and the segment (c_j, c_{j+1}). Since q is on P_F between c_j and c_{j+1}, it must be that q is contained in Q.

Since Q is monotone in the direction from c_j to c_{j+1}, it can be partioned into trapezoids whose top sides are edges of P_j, whose bottom sides are on the line segment (c_j, c_{j+1}), and whose left and right sides are perpendicular to (c_j, c_{j+1}). Refer to Figure 3.5.

Let a and b be the two vertices of P_j that define the trapezoid containing q. We claim that a and b cannot be consecutive on P_j because their Voronoi regions do not share an edge that intersects (c_j, c_{j+1}). We will prove this by showing that in the Voronoi diagram of q, a, and b the bisector of a and b does not intersect the segment (c_j, c_{j+1}). This is sufficient, since this bisector contains the bisector of a and b in the entire Voronoi diagram.

Let C be the circle with center on (c_j, c_{j+1}) and with a and b on its boundary. If the bisector of a and b in the Voronoi diagram of q, a, and b intersects the segment (c_j, c_{j+1}), then C must not contain q. However, C does contain the top, left, right, and bottom sides of the trapezoid containing q. But this can't be, since then C contains the entire trapezoid and contains q. We conclude that there is no point q on the boundary of F between c_j and c_{j+1} that is not on P_j. □

Our c-competitive routing algorithm will visit all the vertices $b_0, \ldots, b_{l-1}$ in order. If b_i and b_{i+1} are consecutive on the Voronoi path (i.e., $b_i = v_j$ and $b_{i+1} = v_{j+1}$ for some j), then our algorithm will use the Voronoi path (i.e., the direct edge) from b_i to b_{i+1}. On the other hand, if b_i and b_{i+1} are not consecutive on the Voronoi path, then by Lemma 3.4, there exists a path from b_i to b_{i+1} of length at most $c_{\text{dfs}} \cdot (\text{x}(b_{i+1}) - \text{x}(b_i))$.

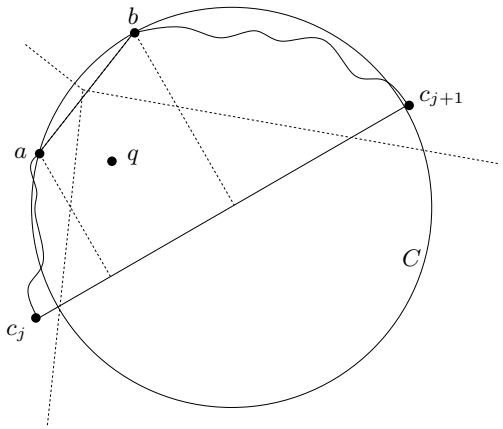


FIG. 3.5. *The proof of Lemma* 3.4.

The difficulty occurs because the algorithm does not know beforehand which path to take. The solution is to simulate exploring both paths "in parallel" and stopping when the first one reaches b_{i+1}.[3]

More formally, let P_V and P_F be defined as in Lemma 3.4. The algorithm for finding a path from b_i to b_{i+1} is described by the following pseudocode.

1: $j \leftarrow 0$, $l_0 \leftarrow \min\{dist(p_0, p_1), dist(q_0, q_1)\}$.
2: **repeat**
3: Explore P_F until reaching b_{i+1} or until reaching a vertex q_x such that $|q_0, \ldots, q_{x+1}| > 2l_j$. If b_{i+1} is reached, then quit; otherwise return to b_i.
4: $j \leftarrow j + 1$, $l_j \leftarrow |q_0, \ldots, q_{y+1}|$.
5: Explore P_V until reaching b_{i+1} or until reaching a vertex p_y such that $|p_0, \ldots, p_{y+1}| > 2l_j$. If b_{i+1} is reached, then quit; otherwise return to b_i.
6: $j \leftarrow j + 1$, $l_j \leftarrow |p_0, \ldots, p_{y+1}|$.
7: **until** b_{i+1} is reached

LEMMA 3.5. *Using the parallel search algorithm described above, a packet reaches* b_{i+1} *after traveling a distance of at most* $9 \cdot c_{\mathrm{dfs}} \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i)) \sim 45.75 \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i))$.

Proof. Clearly the algorithm reaches b_{i+1} in a finite number of steps, since lines 4 and 6 ensure that both paths advance by at least one edge at each iteration. Let k be the maximum value of j, and let d_j be the distance traveled during the jth exploration step of the algorithm. Thus, the total distance d traveled by the packet is given by $d = \sum_{j=0}^{k} d_j$.

Since the algorithm did not terminate with $j = k - 1$, by Lemma 3.4 we have

$$d_k < 2 \cdot c_{\mathrm{dfs}} \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i)). \tag{3.5}$$

Similarly, since the algorithm did not terminate with $j = k - 1$ or $j = k - 2$, we have

$$l_{k-1} < 2 \cdot c_{\mathrm{dfs}} \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i)). \tag{3.6}$$

[3] A similar algorithm for finding an unknown target point on a line is given by Baeza-Yates, Culberson, and Rawlins [1]. See also Klein [12].

Since $l_j \geq 2l_{j-1}$ for each $j > 0$, we have

$$d \leq \sum_{j=0}^{k-1} 2l_j + d_k \tag{3.7}$$

$$\leq \sum_{j=0}^{k-1} 2l_{k-1}/2^j + d_k, \tag{3.8}$$

which immediately yields a bound of $10 \cdot c_{\mathrm{dfs}} \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i))$. To obtain a tighter bound, we note that $d_k > c_{\mathrm{dfs}} \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i))$ implies $l_{k-1} < c_{\mathrm{dfs}} \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i))$. Subject to this constraint, (3.8) is maximized when $l_{k-1} = 2 \cdot c_{\mathrm{dfs}} \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i))$, yielding

$$d \leq \sum_{j=0}^{k-1} 4 \cdot c_{\mathrm{dfs}} \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i))/2^j + c_{\mathrm{dfs}} \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i)) \tag{3.9}$$

$$< 9 \cdot c_{\mathrm{dfs}} \cdot (\mathrm{x}(b_{i+1}) - \mathrm{x}(b_i)). \qquad \square \tag{3.10}$$

Given the positions of v_{dst} and v_{src} the parallel search algorithm described above is easily implemented as part of an $O(1)$ memory routing algorithm. We refer to the combination of the Voronoi routing algorithm with this parallel search algorithm as the *parallel Voronoi routing* algorithm.

THEOREM 3.6. *The parallel Voronoi routing algorithm produces a path whose length is at most* $(9 \cdot c_{\mathrm{dfs}} + \pi/2) \cdot dist(v_{\mathrm{src}}, v_{\mathrm{dst}})$.

Proof. The algorithm incurs two costs: (1) the cost of traveling on subpaths of the Voronoi path that remain above the y-axis, and (2) the cost of applications of the parallel search algorithm. By Lemma 3.3, the first cost is at most $(\pi/2) \cdot dist(v_{\mathrm{src}}, v_{\mathrm{dst}})$. By Lemma 3.5 and the fact that $b_0, \ldots, b_{l-1}$ is x-monotone (Lemma 3.2), the cost of the second is at most $9 \cdot c_{\mathrm{dfs}} \cdot dist(v_{\mathrm{src}}, v_{\mathrm{dst}})$. $\square$

3.3. Empirical results. While it is sometimes possible to come up with pathological examples of triangulations for which an algorithm is not competitive, it is often more reasonable to use the competitive ratio of an algorithm on average or random inputs as an indicator of how it will perform in practice. In this section we describe some experimental results about the competitiveness of our algorithms. All experiments were performed on sets of random points uniformly distributed in the unit square, and each data point is the maximum of 50 independent trials.

The first set of experiments, shown in Figure 3.6, involved measuring the performance of all six routing algorithms on Delaunay triangulations. Compass routing, greedy routing, and Voronoi routing consistently achieve better competitive ratios, with greedy routing slightly worse than the other two. Randomized compass routing, right-hand routing, and parallel Voronoi routing had significantly higher competitive ratios. The results for randomized compass routing and right-hand routing show a significant amount of jitter. This is due to the fact that relatively simple configurations (see Figure 3.1.b) that can easily occur in random point sets result in high competitive ratios for these algorithms. On the other hand, parallel Voronoi routing seems much more stable, and achieves better competitive ratios in practice than its worst-case analysis would indicate.

The most important conclusion drawn from these experiments is that there are no simple configurations (i.e., that occur often in random point sets) that result in

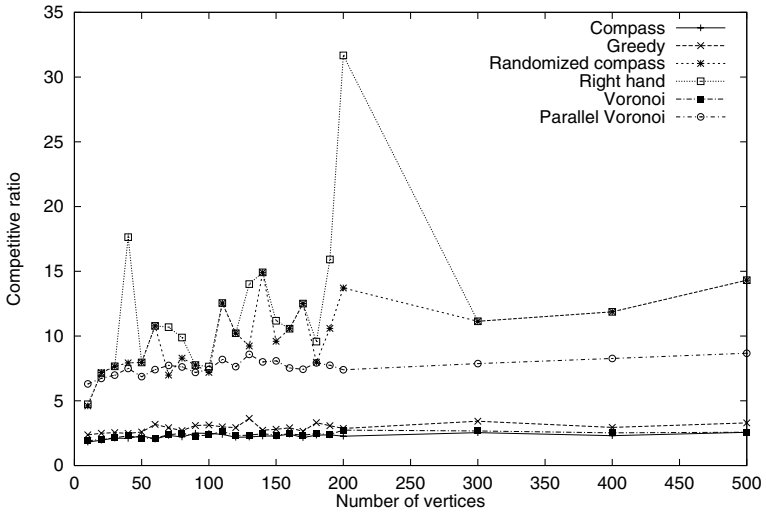


FIG. 3.6. *Empirical competitive ratios for Delaunay triangulations.*

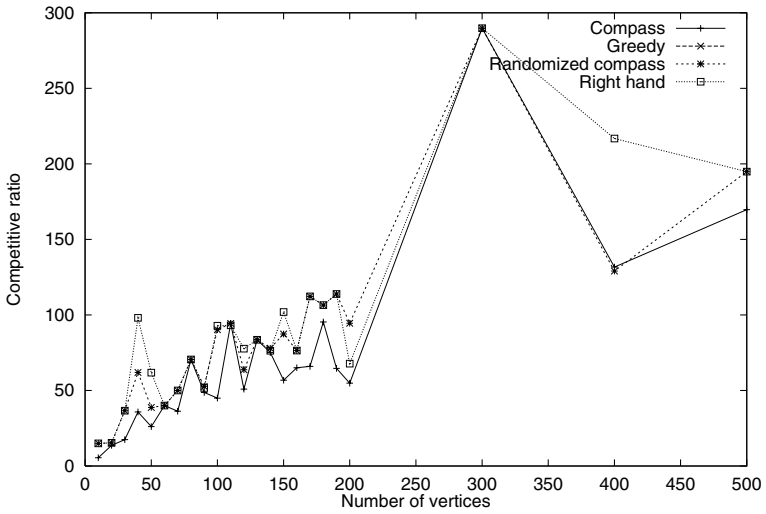


FIG. 3.7. *Empirical competitive ratios for Graham triangulations.*

extremely high competitive ratios for greedy, compass, Voronoi, or parallel Voronoi routing in Delaunay triangulations. This suggests that any of these algorithms would work well in practice.

The four simple routing algorithms of section 2 were also tested on Graham triangulations. These are obtained by first sorting the points by x-coordinate and then triangulating the resulting monotone chain using a linear time algorithm for computing the convex hull of a monotone polygonal chain [17]. The results are shown in Figure 3.7. In these tests it was always the case that at least one of the 50 independent triangulations defeated greedy routing. Thus, there are no results shown for greedy routing. The relative performance of the compass, randomized compass, and

right-hand routing algorithms was the same as for Delaunay triangulations. However, unlike the results for Delaunay triangulations, the competitive ratio appears to be increasing linearly with the number of vertices.

4. Comparison with related work. In this section we survey related work in the area of geometric online routing and compare our results with this work. We restrict our attention to work directly related to routing between the vertices of geometric graphs in which the source and destination are inputs, and we do not consider routing in other geometric settings such as polygons (cf. [9, 10, 12]).

Keil and Gutwin [11] give an algorithm for the construction of a geometric graph called the θ-graph for which a memoryless routing algorithm similar to compass routing always results in a path whose length is at most a constant (dependent only on θ) times the Euclidean distance between v_{src} and v_{dst}.

Kranakis, Singh, and Urrutia [13] study compass routing and provide a proof that no Delaunay triangulation defeats compass routing. The current paper makes use of a very different proof technique to show that compass routing works for a larger class of triangulations. They also describe an $O(1)$ memory routing algorithm that is not defeated by any connected planar graph, thus proving a stronger result than Theorem 2.7.

Lin and Stojmenović [15] and Bose et al. [4] consider online routing in the context of ad hoc wireless networks modeled by unit disk graphs. They provide simulation results for a variety of algorithms that measure success rates (how often a packet never reaches its destination) as well as hop-counts of these algorithms on unit graphs of random point sets.

Lawson's oriented walk [14] is a simple algorithm for point location in Delaunay triangulations without preprocessing. The algorithm can be converted to an $O(1)$ memory routing algorithm that is not defeated by any Delaunay triangulation. The results of the current paper improve on this algorithm by providing two *memoryless* routing algorithms that are not defeated by any Delaunay triangulation.

De Berg et al. [6] describe an algorithm for enumerating all the vertices of a connected planar subdivision using only $O(1)$ additional memory. This algorithm can also be viewed as an $O(1)$ memory routing algorithm. Similarly, in any connected graph with a finite number of vertices, a random walk will eventually visit every vertex. Thus, random walking can be viewed as a randomized memoryless routing algorithm that is not defeated by any graph. Unfortunately paths found by these techniques will usually be much longer than the shortest path, since they are general traversal techniques. In contrast, the right-hand routing and randomized compass routing algorithms make use of information about the source and destination to find more direct paths.

To the best of our knowledge, no literature currently exists on the competitiveness of geometric routing algorithms in our abstract setting, and our parallel Voronoi routing algorithm is the first theoretical result in this area.

5. Conclusions. We have studied the problem of online routing in geometric graphs. Our theoretical results show which types of graphs our algorithms are guaranteed to work on, while our simulation results rank the performance of the algorithms on two types of random triangulations. These results are summarized in Table 5.1.

We conclude with an open problem. In section 2 we showed that no deterministic memoryless routing algorithm works for every 2-connected embedded planar graph. Can a similar argument be made for triangulations, thus proving that randomization or memory is necessary for an algorithm that is not defeated by any triangulation?

TABLE 5.1
Summary of results for greedy routing (GR), compass routing (CR), randomized compass routing (RCR), right-hand routing (RHR), Voronoi routing (VR), and parallel Voronoi routing (PVR) algorithms.

Algorithm	Mem.	Rand.	Class of graphs	Rank 1	Rank 2	Competitive
GR	None	No	Delaunay △'s	3	–	No
CR	None	No	Regular △'s	1	1	No
RCR	None	Yes	All △'s	5	2	No
RHR	$O(1)$	No	Convex subd.	6	3	No
VR	$O(1)$	No	Delaunay △'s	1	–	No
PVR	$O(1)$	No	Delaunay △'s	4	–	Yes

Acknowledgment. The authors would like to thank Silvia Götz for reading and commenting on an earlier version of this paper.

REFERENCES

[1] R. BAEZA-YATES, J. CULBERSON, AND G. RAWLINS, *Searching in the plane*, Inform. and Comput., 106 (1993), pp. 234–252.
[2] J. A. BONDY AND U. S. R. MURTY, *Graph Theory with Applications*, American Elsevier, New York, 1976.
[3] A. BORODIN AND R. EL-YANIV, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.
[4] P. BOSE, P. MORIN, I. STOJMENOVIĆ, AND J. URRUTIA, *Routing with guaranteed delivery in ad hoc wireless networks*, in Proceedings of Discrete Algorithms and Methods for Mobility (DIALM'99), ACM, New York, 1999, pp. 48–55.
[5] V. CHVÁTAL, *A combinatorial theorem in plane geometry*, J. Combin. Theory Ser. B, 18 (1975), pp. 39–41.
[6] M. DE BERG, M. VAN KREVELD, R. VAN OOSTRUM, AND M. OVERMARS, *Simple traversal of a subdivision without extra storage*, Internat. J. Geographic Inform. Systems, 11 (1997), pp. 359–373.
[7] D. DOBKIN, S. J. FRIEDMAN, AND K. J. SUPOWIT, *Delaunay graphs are almost as good as complete graphs*, Discrete Comput. Geom., 5 (1990), pp. 399–407.
[8] H. EDELSBRUNNER, *An acyclicity theorem for cell complexes in d dimension*, Combinatorica, 10 (1988), pp. 251–260.
[9] S. K. GHOSH AND S. SALUJA, *Optimal on-line algorithms for walking with minimum number of turns in unknown streets*, Comput. Geom., 8 (1997), pp. 241–266.
[10] C. ICKING AND R. KLEIN, *Searching for the kernel of a polygon: A competitive strategy*, in Proceedings of the 11th Annual ACM Symposium on Computational Geometry, ACM, New York, 1995, pp. 258–266.
[11] J. M. KEIL AND C. A. GUTWIN, *Classes of graphs which approximate the complete Euclidean graph*, Discrete Comput. Geom., 7 (1992), pp. 13–28.
[12] R. KLEIN, *Walking an unknown street with bounded detour*, Comput. Geom., 1 (1992), pp. 325–351.
[13] E. KRANAKIS, H. SINGH, AND J. URRUTIA, *Compass routing on geometric networks*, in Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99), 1999, pp. 51–54; available online from http://www.cccg.ca/proceedings/1999/.
[14] C. L. LAWSON, *Software for C^1 surface interpolation*, in Mathematical Software III, Academic Press, New York, 1977, pp. 161–194.
[15] X. LIN AND I. STOJMENOVIĆ, *Geographic Distance Routing in Ad Hoc Wireless Networks*, Tech. Report TR-98-10, SITE, University of Ottawa, Canada, 1998.
[16] A. OKABE, B. BOOTS, AND K. SUGIHARA, *Spatial Tesselations: Concepts and Applications of Voronoi Diagrams*, John Wiley and Sons, New York, 1992.
[17] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry*, Springer-Verlag, New York, 1985.
[18] G. M. ZIEGLER, *Lectures on Polytopes*, Grad. Texts in Math. 154, Springer-Verlag, New York, 1994.

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 952–983

DISTRIBUTIONAL RESULTS FOR COSTS OF PARTIAL MATCH QUERIES IN ASYMMETRIC K-DIMENSIONAL TRIES*

WERNER SCHACHINGER†

Abstract. In this paper we study the costs C_N of partial match retrievals in K-dimensional tries (K-d tries), constructed from N records. The probabilistic model that we assume is the asymmetric Bernoulli model: keys are sequences of independently and identically distributed random variables, which assume the values 0 and 1 with probability $p \neq \frac{1}{2}$ and $1-p$, and are pairwise independent. We determine the extremal asymptotic orders that the sequence of expectations $(\mathbb{E}\, C_N)_{N\geq 0}$ may have for different fixed queries, as well as the narrow region that contains $(\mathbb{E}\, C_N)_{N\geq 0}$ for almost every query. Furthermore we show that $(\mathbb{E}\, C_N)_{N\geq 0}$ and $(\operatorname{Var} C_N)_{N\geq 0}$ have the same asymptotics up to a logarithmic factor and, employing a central limit theorem for martingale difference arrays, we prove asymptotic normality of $\frac{C_N - \mathbb{E}\, C_N}{\sqrt{\operatorname{Var} C_N}}$. For random queries, assumed to be independent of the keys and having their specified components distributed according to the same Bernoulli model, no limiting distribution for C_N exists, but we can prove asymptotic normality of $\ln C_N$, when appropriately normalized, and determine $\operatorname{Var} C_N$ up to a logarithmic factor, where now $(\mathbb{E}\, C_N)^2 = o(\operatorname{Var} C_N)$.

Key words. K-d trie, multidimensional data, partial match retrieval, martingale, asymptotic normality

AMS subject classifications. 68W40, 60F05, 68P05, 68P10, 68P20

DOI. 10.1137/S0097539703425873

1. Introduction.

Data structures supporting retrieval of multidimensional data are indispensable for the design of database systems and find further applications in the management of geographical data and the realm of computational geometry. Among the data structures, which have been developed for retrieval of multidimensional data, are Bentley's K-dimensional (K-d) search trees [1], which are comparison based, and Rivest's digital K-d trees [24]. Of the latter type are also the K-d tries (cf. [2, 10, 19]), which will be studied in this paper. Given a set of N records, a particular problem consists in finding all records whose keys match a given query. This means that some fixed components of the K-tuple-keys must coincide with the corresponding components of the query, while the others are not specified and can take arbitrary values. The cost C_N of such a *partial match retrieval* is defined to be the number of the K-d trie's internal nodes visited during the search for all matches.

Partial match retrieval in K-d tries has been studied from the average-case point of view under the Bernoulli and Poisson models of randomness. Flajolet and Puech [10] computed asymptotics (as $N \to \infty$) of $\mathbb{E}\, C_N$ under the symmetric Bernoulli and Poisson models. Kirschenhofer and Prodinger [16] extended these results to K-d digital search trees and Patricia tries. Kirschenhofer, Prodinger, and Szpankowski [17] computed $\mathbb{E}\, C_N$ under the asymmetric Bernoulli model, both for the case of a random query and the case of a fixed periodic query. (No distinction between fixed and random queries has to be made in the symmetric Bernoulli model.) The first result concerning $\operatorname{Var} C_N$ in the symmetric Bernoulli model (the case $K = 2$) can also be found in [17], and, using different methods, the author [25] could settle the case $K \geq 3$. Recently, *relaxed* variants of K-d trees have attracted interest; see [8] and [20]

*Received by the editors April 16, 2003; accepted for publication (in revised form) January 9, 2004; published electronically May 25, 2004.
http://www.siam.org/journals/sicomp/33-4/42587.html
†Department of Statistics and Decision Support Systems, University of Vienna, Brünnerstr. 72, A-1210 Vienna, Austria (Werner.Schachinger@univie.ac.at).

for results concerning expectation and variance of cost. Results regarding limiting distributions have so far only been obtained for the symmetric Bernoulli model [26], where $\frac{C_N - \mathbb{E}\, C_N}{\sqrt{\operatorname{Var} C_N}}$ is shown to converge in distribution and with all its moments to a standard normal random variable. The present paper has its focus on variance and limiting distribution of C_N under the asymmetric Bernoulli model, both for the case of a random query and the case of a fixed query.

For results on partial match retrievals in K-d search trees and quadtrees, ranging from expectations to limiting distributions, we refer to [3, 4, 7, 8, 10, 19, 20, 22, 23].

We start with a description of K-d tries, partial match retrieval, and the probabilistic model under consideration.

1.1. Tries and K-d tries. The trie (cf. [11, 18, 19]) is designed to store data, whose keys are given as sequences over a finite alphabet Σ. Here we confine ourselves to the case $\Sigma = \{0, 1\}$. Now let a set $\mathcal{S} = \{k_i \in \Sigma^{\mathbb{N}} : 1 \le i \le N\}$ of keys be given. The trie built from these keys is a binary tree, whose internal nodes serve as branching nodes. Each leaf (external node) either stores one key or is empty. If we label in this tree each edge to the left (resp., right) 0 (resp., 1), we obtain an encoding of the leaves by taking the 0-1-sequence along the path starting from the root. A key k_i is stored in the leaf encoded by k_i's minimal unique prefix among the N keys in $\mathcal{S}$.

K-d tries (cf. [24, 10]) are built the same way, but some preprocessing of the keys, which are now K-tuples of "simple" keys, has to be done. Given a key $k_i = (k_{i1}, \dots, k_{iK})$, where $k_{i\ell} = (k_{i\ell}^1, k_{i\ell}^2, \dots) \in \Sigma^{\mathbb{N}}$, we construct a key $\tilde{k}_i \in \Sigma^{\mathbb{N}}$ by "shuffling":

$$\tilde{k}_i = (k_{i1}^1, k_{i2}^1, \dots, k_{iK}^1, k_{i1}^2, k_{i2}^2, \dots, k_{iK}^2, \dots).$$

The trie constructed from the keys $\tilde{k}_i$, $1 \le i \le N$, is called a K-d trie. The set of all K-d tries $\mathtt{t}$ built from N keys is denoted by T_N, and $|\mathtt{t}| = N$ is said to be the size of $\mathtt{t}$.

1.2. Partial match retrieval. A *partial match query* is a K-tuple

$$q = (q_1, \dots, q_K) \in \{\Sigma^{\mathbb{N}}, *\}^K.$$

We say that the component q_i is unspecified if $q_i = *$, and is otherwise specified. The task of performing a *partial match retrieval* asks for all keys k_i that match the query q, which means $k_{ij} = q_j$ whenever q_j is specified, and $k_{ij} \in \Sigma^{\mathbb{N}}$ is arbitrary whenever $q_j = *$. The *cost* $C(\mathtt{t}, \tilde{q})$ of the partial match retrieval in a K-d trie $\mathtt{t}$ is defined to be the number of internal nodes of $\mathtt{t}$ visited during the search for the shuffled query $\tilde{q}$ (before shuffling, each component $*$ of q is replaced by the constant sequence $(*, *, *, \dots)$). For any sequence $k = (k^1, k^2, \dots)$ we denote by $k' = (k^2, k^3, \dots)$ the shift to the left by one position of k, and more generally by $k^{(m)}$ the shift to the left by m positions. We can now compute $C(\mathtt{t}, \tilde{q})$ inductively by

$$C(\mathtt{t}, \tilde{q}) = \begin{cases} 0, & |\mathtt{t}| \le 1, \\ 1 + C(\mathtt{t}_\ell, \tilde{q}'), & \tilde{q} = 0\tilde{q}', \\ 1 + C(\mathtt{t}_r, \tilde{q}'), & \tilde{q} = 1\tilde{q}', \\ 1 + C(\mathtt{t}_\ell, \tilde{q}') + C(\mathtt{t}_r, \tilde{q}'), & \tilde{q} = *\tilde{q}', \end{cases} \tag{1}$$

where $\mathtt{t}_\ell$, respectively, $\mathtt{t}_r$, denote the left, respectively, right, subtree of $\mathtt{t}$ when $|\mathtt{t}| \ge 2$.

The *specification pattern* ω of $\tilde{q} = (\tilde{q}_1, \tilde{q}_2, \dots)$ is some element of $\{S, *\}^K$, such that periodic repetition of ω indicates specified and unspecified positions of $\tilde{q}$. The

number of symbols $*$ in a specification pattern $\omega \in \{S, *\}^K$ is denoted by u, and we assume throughout the paper that $0 < u < K$. The set of all $\tilde{q}$ with given specification pattern ω is denoted by K_ω.

1.3. The probabilistic model. We are going to analyze for fixed or random $\boldsymbol{\kappa} \in K_\omega$ and $N \in \mathbb{N}$ the quantity $C(\mathsf{t}, \boldsymbol{\kappa})$, where $\mathsf{t} \in \mathsf{T}_N$ is random and independent of $\boldsymbol{\kappa}$. The probabilistic model we use for T_N assumes that the shuffled keys $\tilde{k}_i$ are independently and identically distributed (i.i.d.) sequences of elements from Σ, where 0 and 1 occur with probability $p \in]0,1[$ and $q := 1 - p$, respectively, and that keys $\tilde{k}_i, \tilde{k}_j$ are independent for $1 \le i < j \le N$. If $p = \frac{1}{2}$, this is called the symmetric, otherwise the asymmetric, Bernoulli model. These models give rise to a sequence of discrete probability spaces $(\mathsf{T}_N, \mathcal{T}_N, P_N)$, with $\mathcal{T}_N$ simply the set of subsets of T_N. (Regarding discreteness, there are indeed only finitely many binary trees of fixed height, and the minimal prefixes needed for the construction of the trie are almost surely finite. On the other hand, T_N is infinite for $N \ge 2$, due to the fact that leaves can be empty.) The computation of probabilities $P_{|\mathsf{t}|}(\mathsf{t})$ is facilitated using the binomial splitting probabilities

$$p_{N,k} := \mathbb{P}\left(|\mathsf{t}_\ell| = k \big| |\mathsf{t}| = N\right) = \binom{N}{k} p^k q^{N-k}.$$

Both T_0 and T_1 are singletons; therefore $P_0(\mathsf{t}) = P_1(\mathsf{t}) = 1$ for t taken from T_0, respectively, T_1. For $|\mathsf{t}| \ge 2$, $\mathsf{t} = \overset{\mathrm{O}}{/\ \backslash}_{\mathsf{t}_l\ \mathsf{t}_r}$, we have $P_{|\mathsf{t}|}(\mathsf{t}) = p_{|\mathsf{t}|,|\mathsf{t}_\ell|} P_{|\mathsf{t}_\ell|}(\mathsf{t}_\ell) P_{|\mathsf{t}_r|}(\mathsf{t}_r)$.

We further introduce the probability space $(K_\omega, \mathcal{B}(K_\omega), Q)$, where, according to Q, the specified elements of $\boldsymbol{\kappa} \in K_\omega$ constitute an i.i.d. sequence of elements from Σ, where 0 and 1 occur with the probabilities p and q, just as for the keys.

The main objects of our study are the random variables $C_N := C(\mathsf{t}, \boldsymbol{\kappa})$ defined on $\mathsf{T}_N \times K_\omega$, and $F_{\boldsymbol{\kappa},N} = F_{\boldsymbol{\kappa},N}(\mathsf{t}) := C(\mathsf{t}, \boldsymbol{\kappa})$ defined on T_N, as well as $F^i_{\boldsymbol{\kappa},N} = F^i_{\boldsymbol{\kappa},N}(\mathsf{t}) := C(\mathsf{t}, \boldsymbol{\kappa}^{(i)})$ for $i \ge 0$, defined on T_N, where $F^0_{\boldsymbol{\kappa},N} = F_{\boldsymbol{\kappa},N}$. For some statements we need the sequences of random variables $(C_N)_{N\ge0}$, respectively, $(F^i_{\boldsymbol{\kappa},N})_{N\ge0}$, live on the same probability space, which we choose to be $(\Sigma^{\mathbb{N}\times\mathbb{N}} \times K_\omega, \mathcal{B}(\Sigma^{\mathbb{N}\times\mathbb{N}} \times K_\omega), P \times Q)$, respectively, $(\Sigma^{\mathbb{N}\times\mathbb{N}}, \mathcal{B}(\Sigma^{\mathbb{N}\times\mathbb{N}}), P)$, with $\boldsymbol{\tau} = (k_i)_{i\in\mathbb{N}} \in \Sigma^{\mathbb{N}\times\mathbb{N}}$ representing an infinite sequence of keys, where, according to P, all the symbols of all the keys form an i.i.d. sequence of elements from Σ, where 0 and 1 occur with probability p and q, respectively. We denote by $\boldsymbol{\tau}_N = (k_i)_{i=1}^N$ the set of the first N keys and by $\mathsf{t}(\boldsymbol{\tau}_N)$ the trie built from that set. Costs for random and fixed queries are then defined by $C_N = C_N(\boldsymbol{\tau}, \boldsymbol{\kappa}) = C(\mathsf{t}(\boldsymbol{\tau}_N), \boldsymbol{\kappa})$ and $F^i_{\boldsymbol{\kappa},N} = F^i_{\boldsymbol{\kappa},N}(\boldsymbol{\tau}) = C(\mathsf{t}(\boldsymbol{\tau}_N), \boldsymbol{\kappa}^{(i)})$. By definition, the random sequence $(F^i_{\boldsymbol{\kappa},N})_{N\ge0}$ is increasing. We let $f^i_{\boldsymbol{\kappa},N} := \mathbb{E}\, F^i_{\boldsymbol{\kappa},N}$ and $v^i_{\boldsymbol{\kappa},N} := \operatorname{Var} F^i_{\boldsymbol{\kappa},N}$ for $i \ge 0$ and use the shorthand notation $f_{\boldsymbol{\kappa},N} = f^0_{\boldsymbol{\kappa},N}$, $v_{\boldsymbol{\kappa},N} = v^0_{\boldsymbol{\kappa},N}$. The subscript $\boldsymbol{\kappa}$ will frequently be suppressed.

Throughout the paper we denote convergence (resp., equality) in distribution by $\xrightarrow{\mathcal{D}}$ (resp., $\overset{\mathcal{D}}{=}$). We write $X \sim F$ if the random variable X has distribution F, and $\mathcal{N}(0,1)$ denotes a standard normal random variable. We put $a \vee b = \max(a,b)$ and $a \wedge b = \min(a,b)$ for any real numbers a and b. The indicator function of a set A is denoted by $\mathbb{1}_A$, and for a Boolean expression B we let $\mathbb{1}_{\{B\}}$ be 1 if B is true and 0 otherwise. The difference operator, Δ, is defined by $\Delta x_k = x_{k+1} - x_k$. Bold lowercase letters, such as $\boldsymbol{\kappa}$ or $\mathbf{f}^i$, always denote sequences, and bold uppercase letters are used for matrices. We will use the standard asymptotic notation $\mathcal{O}$, o, Ω, and Θ.

We use the symbol $\sim$ also to denote asymptotic equivalence, and write occasionally $a_n \asymp b_n$ if $a_n = \Theta(b_n)$.

2. Main results. Our first result, Theorem 1, and its Corollary 1, demonstrate that in the asymmetric Bernoulli model the expectations $f_{\boldsymbol{\kappa},N}$ may enjoy a growth between $\Omega(N^{\gamma_2})$ and $\mathcal{O}(N^{\gamma_1})$ and will almost surely be "close" to N^{β} for certain constants $\gamma_2 < \beta < \gamma_1$. Note that things are different in the symmetric Bernoulli model, where $F_{\boldsymbol{\kappa},N} \stackrel{\mathcal{D}}{=} C_N$, and hence $f_{\boldsymbol{\kappa},N} = \mathbb{E}\, C_N$ holds regardless of the value of the query $\boldsymbol{\kappa}$, and where $\mathbb{E}\, C_N = \Theta(N^{u/K})$ was shown to hold in [10], with a positive periodic function of $\ln N$ hidden in the Θ symbol.

THEOREM 1. *Let $p \in [\frac{1}{2}, 1[$, $q := 1-p$, $0 < u < K$ and define $\beta = \beta(p)$, $\gamma_1 = \gamma_1(p)$, and $\gamma_2 = \gamma_2(p)$ to be the unique real solutions of the equations*

$$(p^p q^q)^{(K-u)\beta}(p^\beta + q^\beta)^u = 1, \quad p^{(K-u)\gamma_1}(p^{\gamma_1} + q^{\gamma_1})^u = 1, \quad q^{(K-u)\gamma_2}(p^{\gamma_2} + q^{\gamma_2})^u = 1.$$

Moreover denote $P_x := \frac{p^x}{p^x + q^x}$ and $Q_x := 1 - P_x$ for $x \in \mathbb{R}$. Then

$$\begin{aligned} \sup_{\boldsymbol{\kappa} \in K_\omega} f_{\boldsymbol{\kappa},N} &= \Theta(N^{\gamma_1}), \\ \inf_{\boldsymbol{\kappa} \in K_\omega} f_{\boldsymbol{\kappa},N} &= \Theta(N^{\gamma_2}). \end{aligned} \tag{2}$$

Furthermore, for Q-almost every $\boldsymbol{\kappa} \in K_\omega$ we have

$$\lim_{N\to\infty} \frac{|\ln f_{\boldsymbol{\kappa},N} - \beta \ln N - \beta S_{\boldsymbol{\kappa},\lfloor c_\beta \ln N \rfloor}|}{\ln^{\frac{1}{3}} N \ln\ln N} = 0, \tag{3}$$

where

$$S_{\boldsymbol{\kappa},n} := \sum_{j=1}^{Kn} \big(\mathbb{1}_{\{\kappa_j = 0\}} \ln p + \mathbb{1}_{\{\kappa_j = 1\}} \ln q \big) - n(K-u) \ln p^p q^q$$

and

$$c_\beta := \frac{\beta}{u \ln\Big(P_\beta^{-P_\beta} Q_\beta^{-Q_\beta} \Big)}.$$

It is important to note that with

$$X_{\boldsymbol{\kappa},j} := \sum_{i=jK-K+1}^{jK} \big(\mathbb{1}_{\{\kappa_i = 0\}} \ln p + \mathbb{1}_{\{\kappa_i = 1\}} \ln q \big) - (K-u) \ln p^p q^q$$

we have $S_{\boldsymbol{\kappa},n} = \sum_{j=1}^{n} X_{\boldsymbol{\kappa},j}$. Note further that with respect to Q, the random variables $(X_{\boldsymbol{\kappa},j})_{j\geq 1}$ are i.i.d. with $\mathbb{E}\, X_{\boldsymbol{\kappa},1} = 0$ and $\operatorname{Var} X_{\boldsymbol{\kappa},1} = (K-u)pq \ln^2 \frac{p}{q}$. In fact, $X_{\boldsymbol{\kappa},1}$ has the distribution of $\ln \frac{p}{q}$ times a centered binomially $B(K-u,p)$ distributed random variable. Thus the following corollary is a simple consequence of the law of the iterated logarithm and of the central limit theorem for sums of i.i.d. random variables with finite variance.

COROLLARY 1. *For Q-almost every $\boldsymbol{\kappa} \in K_\omega$ we have*

$$\begin{aligned} \limsup_{N\to\infty} \frac{\ln f_{\boldsymbol{\kappa},N} - \beta \ln N}{\sqrt{2\beta \ln N \ln\ln\ln N}} &= \sigma, \\ \liminf_{N\to\infty} \frac{\ln f_{\boldsymbol{\kappa},N} - \beta \ln N}{\sqrt{2\beta \ln N \ln\ln\ln N}} &= -\sigma, \end{aligned} \tag{4}$$

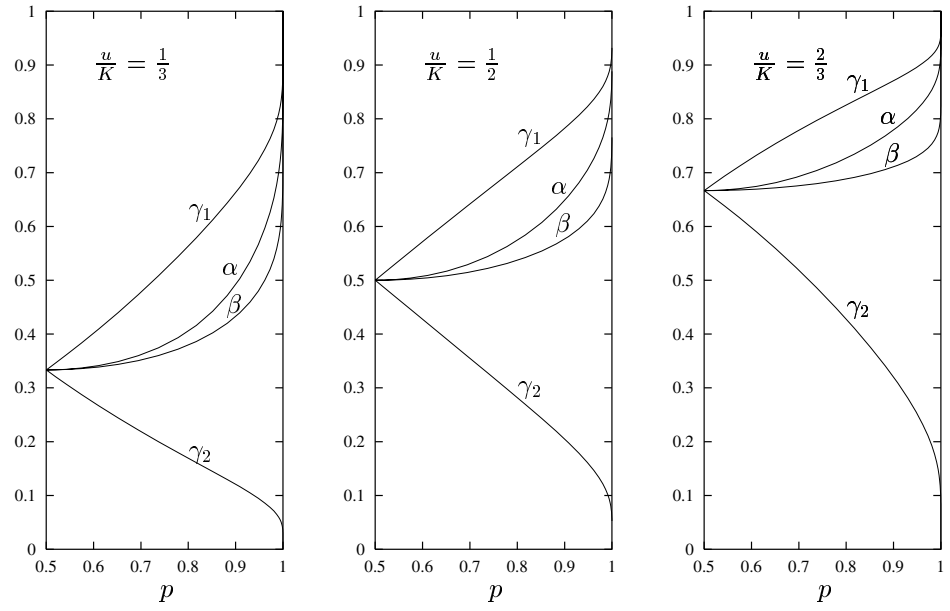


FIG. 1. *The exponents γ_1, γ_2, and β from Theorem* 1 *and α from Remark* 2 *for various values of $\frac{u}{K}$; cf. also the table in* [17].

where $\sigma = \sigma(p) = \beta \ln \frac{p}{q} \sqrt{\left(\frac{K}{u} - 1\right) pq / \ln\left(P_\beta^{-P_\beta} Q_\beta^{-Q_\beta}\right)}$. *Moreover*

$$\frac{\ln f_{\boldsymbol{\kappa},N} - \beta \ln N}{\sqrt{\beta \ln N}} \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma^2). \tag{5}$$

Remark 1. Note that $0 < \gamma_2 < \beta < \gamma_1 < 1$, unless $p = \frac{1}{2}$, in which case $\gamma_2 = \beta = \gamma_1 = \frac{u}{K}$; cf. Figure 1. For an illustration of (3), see Figure 2.

Remark 2. It follows from [17, Theorem 1.1] that we have $N^{-\alpha} \mathbb{E}\, f_{\boldsymbol{\kappa},N} = \Theta(1)$, where the expectation is with respect to Q, and where $\alpha = \alpha(p)$ is the unique real solution of the equation

$$(p^{1+\alpha} + q^{1+\alpha})^{K-u} (p^\alpha + q^\alpha)^u = 1.$$

We have $\beta(p) \le \alpha(p)$ with equality only for $p \in \{\frac{1}{2}, 1\}$: $\beta(\frac{1}{2}) = \alpha(\frac{1}{2}) = \frac{u}{K}$ and $\lim_{p\to 1} \beta(p) = \lim_{p\to 1} \alpha(p) = 1$.

Remark 3. Theorem 1 also holds true for $u = K$, in which case $K_\omega = \{(*, *, \dots)\}$ is a singleton, and we have $\gamma_1 = \beta = \gamma_2 = 1$ and $\sigma = 0$. In the case $p = \frac{1}{2}$ the random variable $f_{\boldsymbol{\kappa},N}$ is constant on K_ω, and again $\sigma = 0$. For $0 < u < K$ fixed, $\sigma(p)$ appears to be strictly increasing with p (we have not proved that), and a tedious calculation shows that $\lim_{p\to 1} \sigma(p) = 1$.

The next two theorems deal with variances. It is known from [17, 25] that in the symmetric Bernoulli model $\operatorname{Var} C_N = \Theta(N^{u/K})$ holds, again with a positive periodic function of $\ln N$ hidden in the Θ symbol, which implies $\operatorname{Var} C_N = \mathcal{O}(\mathbb{E}\, C_N)$. As it will turn out in Theorem 2, that property is shared, up to a logarithmic factor, by the asymmetric fixed query model. On the other hand, Theorem 3 implies $(\mathbb{E}\, C_N)^2 = o(\operatorname{Var} C_N)$ for the asymmetric random query model. Finally, Theorem 4 shows that also the central limit theorem known for the symmetric Bernoulli model (see [26]) persists in the asymmetric fixed query model.

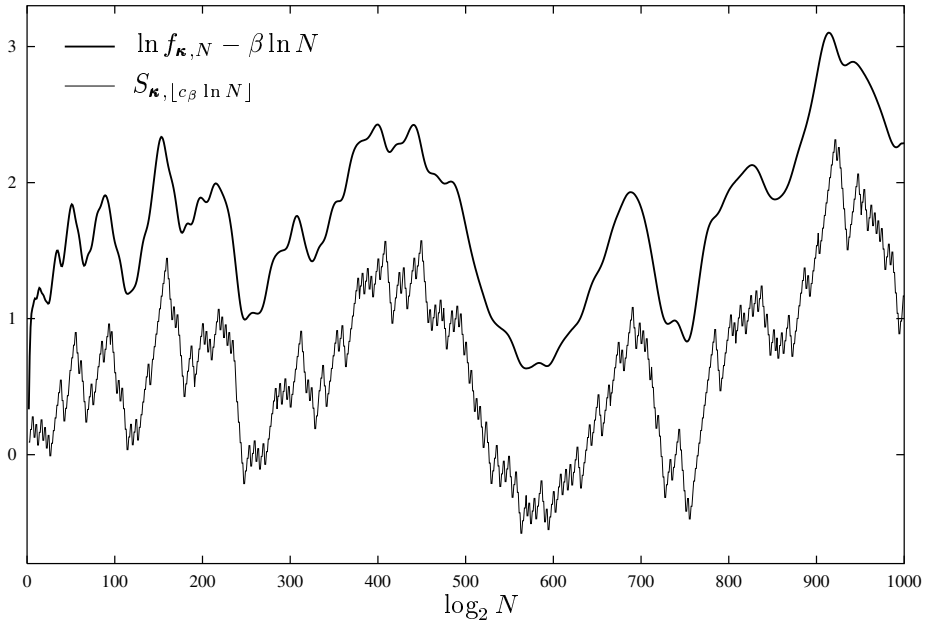


FIG. 2. *An illustration of* (3) *for a particular* $\kappa \in K_\omega$, *with* $\omega = (*, S)$, $p = \frac{1}{2}(\sqrt{5}-1)$, *and* $N \le 2^{1000}$. *The difference between the two curves seems to be bounded, as* $N \to \infty$, *but actually it is not; cf. Remark* 7.

THEOREM 2. *Uniformly in* $\boldsymbol{\kappa} \in K_\omega$, *the variance* $v_{\boldsymbol{\kappa},N}$ *satisfies*

$$v_{\boldsymbol{\kappa},N} = \begin{cases} \mathcal{O}(f_{\boldsymbol{\kappa},N} \ln N), \\ \Omega(f_{\boldsymbol{\kappa},N}). \end{cases}$$

Remark 4. The stronger upper estimate $v_{\boldsymbol{\kappa},N} = \mathcal{O}(f_{\boldsymbol{\kappa},N})$ follows from results obtained in [27] by means of poissonization techniques.

COROLLARY 2. *If in Corollary* 1 *we replace* $f_{\boldsymbol{\kappa},N}$ *by* C_N, *then* (4) *holds* $P \times Q$-*almost surely, and also* (5) *remains valid.*

Moreover all the limit laws that we can obtain for $\frac{C_N - b_N}{a_N}$ *by choosing appropriate normalizing sequences* $(a_N)_{N\ge 0}, (b_N)_{N\ge 0}$ *are degenerate.*

THEOREM 3. *Let* $\frac{1}{2} < p < 1$, *and let* $\xi = \xi(p)$ *be defined as the unique real solution of the equation*

$$(p^{1+2\xi} + q^{1+2\xi})^{(K-u)/2}(p^\xi + q^\xi)^u = 1.$$

Then $\alpha < \xi < \gamma_1$ *and, for large* N,

$$\operatorname{Var} C_N = N^{2\xi} e^{\mathcal{O}(\ln \ln N)}.$$

THEOREM 4. *We have for every fixed* $\boldsymbol{\kappa} \in K_\omega$

$$\frac{F_{\boldsymbol{\kappa},N} - f_{\boldsymbol{\kappa},N}}{\sqrt{v_{\boldsymbol{\kappa},N}}} \xrightarrow{\mathcal{D}} \mathcal{N}(0,1).$$

The paper is organized as follows. Section 3 starts with some preliminaries and continues with the proof of Theorem 1. Several lemmas will be needed. In particular, the close coupling between $f_{\boldsymbol{\kappa},N}$ and a certain sequence of partial sums derived

from $\boldsymbol{\kappa}$, which is expressed in (3), will become a corollary of Lemma 4, which allows for extensions of Theorem 1 and Corollary 1 to models, where the query is still independent of the keys but need not follow the same Bernoulli model as the keys. In section 4 we give proofs of the results concerning variances. Two lemmas used for establishing the Ω estimate in Theorem 2 might be of independent interest. In the proof of Theorem 3 inverse Mellin integrals of functions with dominant algebraic singularities have to be calculated. Finally in section 5 we prove Theorem 4 by invoking a central limit theorem for martingale difference arrays, and section 6 concludes the paper.

3. The expectation.

3.1. Some preliminaries and notation. For $i \in \mathbb{N}$ we denote by $\nu(i)$ the string obtained by stripping off the leading 1 in i's binary representation, i.e., $\nu(i) := s'$, when $i = (s)_2$. For any binary tree t we denote by t^i the subtree of t which has its root in the node v_i, which we define to be the node to which we are guided by the string $\nu(i)$. (If v_i is not in the vertex set of t, then t^i is empty.) For example, v_1 is the root of t (thus $\mathsf{t}^1 = \mathsf{t}$, $\mathsf{t}^2 = \mathsf{t}_\ell$, and $\mathsf{t}^3 = \mathsf{t}_r$), and we are guided to v_5 by the string $\nu(5) = 01$, since $5 = (101)_2$.

It readily follows from (1) and properties of the Bernoulli model that for fixed $\boldsymbol{\kappa} \in K_\omega$ the random variables $F^i_{\boldsymbol{\kappa},N}$ satisfy, for $i, N \geq 0$,

$$F^i_{\boldsymbol{\kappa},N} \stackrel{\mathcal{D}}{=} \mathbb{1}_{\{N\geq 2\}} \left(1 + a_{0,i+1} F^{i+1}_{\boldsymbol{\kappa},k} + a_{1,i+1} \bar{F}^{i+1}_{\boldsymbol{\kappa},N-k}\right), \tag{6}$$

where k follows a binomial distribution $B(N,p)$, the sequence $(\bar{F}^{i+1}_{\boldsymbol{\kappa},\ell})_{\ell\geq 0}$ is an independent copy of $(F^{i+1}_{\boldsymbol{\kappa},\ell})_{\ell\geq 0}$, and finally

$$a_{0,i} = \mathbb{1}_{\{\kappa_i \in \{0,*\}\}} \quad \text{and} \quad a_{1,i} = \mathbb{1}_{\{\kappa_i \in \{1,*\}\}}.$$

Indeed, $(F^i_{\boldsymbol{\kappa},N})_{i,N\geq 0}$ can be defined by (6), i.e., $(X^i_N)_{i,N\geq 0} = (F^i_{\boldsymbol{\kappa},N})_{i,N\geq 0}$ is the only finite solution, up to equality in distribution, to

$$X^i_N \stackrel{\mathcal{D}}{=} \mathbb{1}_{\{N\geq 2\}} \left(1 + a_{0,i+1} X^{i+1}_k + a_{1,i+1} \bar{X}^{i+1}_{N-k}\right)$$

with the same assumptions on distributions and independence as in (6). To see that, we iterate the latter equation $M-1$ times, thus representing X^0_N as

$$X^0_N \stackrel{\mathcal{D}}{=} \sum_{m=1}^{2^M-1} \varepsilon^0_m \mathbb{1}_{\{|\mathsf{t}^m|\geq 2\}} + \sum_{m=2^M}^{2^{M+1}-1} \varepsilon^0_m \mathbb{1}_{\{|\mathsf{t}^m|\geq 2\}} X^{M,m}_{|\mathsf{t}^m|}, \tag{7}$$

where

$$\varepsilon^i_m = \mathbb{1}_{\{\nu(m) \text{ matches } \boldsymbol{\kappa}^{(i)}\}}. \tag{8}$$

(We say that a finite string s matches a query $\boldsymbol{\kappa}$ if s is a prefix of some infinite string $\bar{s}$ that matches $\boldsymbol{\kappa}$.) Moreover $X^{M,m}_k \stackrel{\mathcal{D}}{=} X^M_k$ for $2^M \leq m < 2^{M+1}$, and $(X^{M,m}_k)_{k\geq 0}$ and $(X^{M,m'}_k)_{k\geq 0}$ are independent for $2^M \leq m < m' < 2^{M+1}$. Now, the probability that the second sum of (7) equals 0 is not smaller than $\mathbb{P}(\mathsf{height}(\mathsf{t}) \leq M)$, which approaches 1 as $M \to \infty$; cf. [19, p. 257] for the limiting distribution of the height of a binary asymmetric trie. This proves that the system of equations (6) indeed

uniquely determines $(F^i_{\boldsymbol{\kappa},N})_{i,N\ge 0}$. As a byproduct we have derived the almost surely convergent representation

$$F^i_{\boldsymbol{\kappa},N} = \sum_{m\ge 1} \varepsilon^i_m \mathbb{1}_{\{|\mathtt{t}^m|\ge 2\}}, \tag{9}$$

where ε^i_m depends only on $\boldsymbol{\kappa}$, and $\mathbb{1}_{\{|\mathtt{t}^m|\ge 2\}}$ depends only on $\mathtt{t}$. Note that the random variables summed in (9) are not independent. Equation (6) could be used to prove that moments of all orders of $F_{\boldsymbol{\kappa},N}$ exist. This, however, follows more easily from the fact that $F_{\boldsymbol{\kappa},N}$ is not larger than I_N, the number of internal nodes of the corresponding trie, which has itself moments of all orders (cf. [15]).

Proof of Theorem 1. We will actually show that the sup (and likewise the inf) in (2) is attained for each N, and it is attained at the same $\boldsymbol{\kappa}$ for all N. Denoting $\mathbf{f}^i_{\boldsymbol{\kappa}} = (f^i_{\boldsymbol{\kappa},N})_{N\ge 0}$, we derive from (6)

$$\mathbf{f}^i_{\boldsymbol{\kappa}} = \mathbf{e} + a_{0,i+1}\mathbf{M}_p\mathbf{f}^{i+1}_{\boldsymbol{\kappa}} + a_{1,i+1}\mathbf{M}_q\mathbf{f}^{i+1}_{\boldsymbol{\kappa}} = \mathbf{e} + \mathbf{B}_{i+1}\mathbf{f}^{i+1}_{\boldsymbol{\kappa}}, \tag{10}$$

where $\mathbf{e} := (0,0,1,1,1,\dots)$, $f^i_{\boldsymbol{\kappa},0} = f^i_{\boldsymbol{\kappa},1} = 0$ for $i \ge 0$, the matrix $\mathbf{M}_r$ is defined by $(\mathbf{M}_r)_{N,k} := \binom{N}{k} r^k(1-r)^{N-k}$ (with $0^0 := 1$) and has the property $\mathbf{M}_r\mathbf{M}_{r'} = \mathbf{M}_{rr'} = \mathbf{M}_{r'}\mathbf{M}_r$, and

$$\mathbf{B}_i := a_{0,i}\mathbf{M}_p + a_{1,i}\mathbf{M}_q = \begin{cases} \mathbf{M}_p, & \kappa_i = 0,\\ \mathbf{M}_q, & \kappa_i = 1,\\ \mathbf{M}_p + \mathbf{M}_q, & \kappa_i = *. \end{cases}$$

Iterating (10), we obtain

$$\begin{aligned} \mathbf{f}_{\boldsymbol{\kappa}} = (\mathbf{I} + \mathbf{B}_1 + \mathbf{B}_1\mathbf{B}_2 + \mathbf{B}_1\mathbf{B}_2\mathbf{B}_3 + \cdots)\mathbf{e} &= \sum_{n\ge 0}\prod_{i=1}^{n}\mathbf{B}_i\mathbf{e},\\ &= \sum_{n\ge 0}(\mathbf{M}_p + \mathbf{M}_q)^{\mu_n}\mathbf{M}_{\rho_n}\mathbf{e}, \end{aligned} \tag{11}$$

where $\prod_{i=1}^{0}\mathbf{B}_i = \mathbf{I}$ is the infinite identity matrix, and

$$\mu_n = \mu_n(\boldsymbol{\kappa}) = \sum_{i=1}^{n}\mathbb{1}_{\{\kappa_i = *\}} = \sum_{i=1}^{n} a_{0,i}a_{1,i}, \tag{12}$$

$$\rho_n = \rho_n(\boldsymbol{\kappa}) = \prod_{i=1}^{n}\left(p\mathbb{1}_{\{\kappa_i\in\{0,*\}\}} + q\mathbb{1}_{\{\kappa_i\in\{1,*\}\}}\right) = \prod_{i=1}^{n}(pa_{0,i} + qa_{1,i}). \tag{13}$$

Several lemmas are needed in the further investigation of (11). In the following we write $\mathbf{x}\nearrow$ if the sequence $\mathbf{x}$ is increasing, and we define a partial order $\preceq$ on sequences by

$$\mathbf{x}\preceq\mathbf{y} \;:\Leftrightarrow\; x_N \le y_N \quad \text{for all } N.$$

First we cite a lemma from [27], listing some properties of the matrices $\mathbf{M}_r$.

LEMMA 1. *For $0 \le r \le 1$ and any sequence $\mathbf{x}$ we have the following:*

1. $\mathbf{x} \succeq 0 \;\Rightarrow\; \mathbf{M}_r\mathbf{x} \succeq 0$.
2. $\Delta\mathbf{M}_r\mathbf{x} = r\mathbf{M}_r\Delta\mathbf{x}$.
3. $\mathbf{x} \nearrow \Rightarrow\; \mathbf{x} \succeq \mathbf{M}_r\mathbf{x}$ *and* $\mathbf{M}_r\mathbf{x} \nearrow$.
4. *If* $\mathbf{x} \succeq 0$ *and* $|\Delta^m x_N| \le c\frac{x_{N+m}}{(N+m)^{\underline{m}}}$ *for some* $c > 0$, *some integer* $m \ge 1$, *and all* $N \ge 0$, *then* $|\Delta^m(\mathbf{M}_r\mathbf{x})_N| \le c\frac{(\mathbf{M}_r\mathbf{x})_{N+m}}{(N+m)^{\underline{m}}}$ *for all* $N \ge 0$, *where* $n^{\underline{m}} = n(n-1)\cdot\ldots\cdot(n-m+1)$ *denotes falling factorial powers.*

A few simple facts concerning the sequences $\mathbf{f}^i$ are listed in the next lemma.

LEMMA 2. *For $i \ge 0$ and any fixed $\boldsymbol{\kappa} \in K_\omega$ (which we suppress) we have the following:*

1. $f_N^i \ge f_M^i \ge \frac{M(M-1)}{N(N-1)} f_N^i$ *for* $2 \le M \le N$, *in particular* $\mathbf{f}^i \nearrow$.
2. $f_N^i = \Theta\left(f_N^{i+1}\right)$, *as* $N \to \infty$.

Proof. Since we can represent $\mathbf{f}^i$ as

$$\mathbf{f}^i = \sum_{n\ge 0}\prod_{k=1}^{n}\mathbf{B}_{i+k}\mathbf{e} = \sum_{k\ge 0} b_k\mathbf{M}_{p_k}\mathbf{e}, \tag{14}$$

with $0 < p_k \le 1$ and $b_k > 0$ for $k \ge 0$, and since $\mathbf{e} \nearrow$, the left inequality of the first fact follows from property 3 of Lemma 1. Denoting for $0 < x \le 1$

(15)

$$E_N(x) := (\mathbf{M}_x\mathbf{e})_N = 1 - (1-x)^N - Nx(1-x)^{N-1} = N(N-1)\int_0^x \xi(1-\xi)^{N-2}d\xi,$$

we deduce $E_M(x) \ge \frac{M(M-1)}{N(N-1)}E_N(x)$, which proves the right inequality of the first fact. Furthermore, from (10) and $\mathbf{f}^{i+1} \succeq \mathbf{e}$ we deduce $f_N^i \le 1 + 2f_N^{i+1} \le 3f_N^{i+1}$, and, since $\mathbf{f}^{i+1} \nearrow$ and $(\mathbf{M}_p + \mathbf{M}_q)\mathbf{f}^{i+1} \succeq \mathbf{M}_p\mathbf{f}^{i+1} \succeq \mathbf{M}_q\mathbf{f}^{i+1} = \mathbf{M}_{q/p}\mathbf{M}_p\mathbf{f}^{i+1}$ by property 3 of Lemma 1,

$$f_N^i \ge 1 + \sum_{k=0}^{N}\binom{N}{k}q^k p^{N-k} f_k^{i+1} \ge \sum_{k=0}^{N}\binom{N}{k}q^k p^{N-k}\frac{k(k-1)}{N(N-1)}f_N^{i+1} = q^2 f_N^{i+1},$$

which completes the proof. ∎

A key step in the proof of (2) is the following result.

LEMMA 3. *With $\rho_n(\boldsymbol{\kappa})$ defined in* (13), *we let $\boldsymbol{\rho}(\boldsymbol{\kappa}) = (\rho_n(\boldsymbol{\kappa}))_{n\ge 0}$. If $\boldsymbol{\kappa}, \bar{\boldsymbol{\kappa}} \in K_\omega$ satisfy*

$$\boldsymbol{\rho}(\boldsymbol{\kappa}) \succeq \boldsymbol{\rho}(\bar{\boldsymbol{\kappa}}),$$

then

$$\mathbf{f}_{\boldsymbol{\kappa}} \succeq \mathbf{f}_{\bar{\boldsymbol{\kappa}}}.$$

Proof. Since μ_n depends only on the specification pattern ω, we have $\mu_n(\boldsymbol{\kappa}) = \mu_n(\bar{\boldsymbol{\kappa}})$ for $n \ge 0$. Using (11), we derive

$$\begin{aligned}\mathbf{f}_{\boldsymbol{\kappa}} - \mathbf{f}_{\bar{\boldsymbol{\kappa}}} &= \sum_{n\ge 0}(\mathbf{M}_p + \mathbf{M}_q)^{\mu_n}\mathbf{M}_{\rho_n(\boldsymbol{\kappa})}\mathbf{e} - \sum_{n\ge 0}(\mathbf{M}_p + \mathbf{M}_q)^{\mu_n}\mathbf{M}_{\rho_n(\bar{\boldsymbol{\kappa}})}\mathbf{e}\\ &= \sum_{n\ge 0}\left(\mathbf{I} - \mathbf{M}_{\rho_n(\bar{\boldsymbol{\kappa}})/\rho_n(\boldsymbol{\kappa})}\right)\mathbf{M}_{\rho_n(\boldsymbol{\kappa})}(\mathbf{M}_p + \mathbf{M}_q)^{\mu_n}\mathbf{e} \succeq 0,\end{aligned}$$

where in the latter sum each term is $\succeq 0$ by property 3 of Lemma 1. ∎

Next we note that there are unique sequences $\boldsymbol{\kappa}^{\sup} \in K_\omega \cap \{0,*\}^{\mathbb{N}}$ and $\boldsymbol{\kappa}^{\inf} \in K_\omega \cap \{1,*\}^{\mathbb{N}}$, such that $\boldsymbol{\rho}(\boldsymbol{\kappa}^{\sup}) \succeq \boldsymbol{\rho}(\boldsymbol{\kappa}) \succeq \boldsymbol{\rho}(\boldsymbol{\kappa}^{\inf})$ holds for $\boldsymbol{\kappa} \in K_\omega$, to which Lemma 3 applies, yielding

$$\sup_{\boldsymbol{\kappa}\in K_\omega} \mathbf{f}_{\boldsymbol{\kappa}} = \mathbf{f}_{\boldsymbol{\kappa}^{\sup}} = \sum_{n\geq 0} (\mathbf{M}_p + \mathbf{M}_q)^{\mu_n} \mathbf{M}_{p^{n-\mu_n}} \mathbf{e} =: \mathbf{f}^{\sup},$$

and similarly

$$\inf_{\boldsymbol{\kappa}\in K_\omega} \mathbf{f}_{\boldsymbol{\kappa}} = \mathbf{f}_{\boldsymbol{\kappa}^{\inf}} = \sum_{n\geq 0} (\mathbf{M}_p + \mathbf{M}_q)^{\mu_n} \mathbf{M}_{q^{n-\mu_n}} \mathbf{e} =: \mathbf{f}^{\inf}.$$

To complete the proof of (2) we have to derive Θ-estimates of the sequences $\mathbf{f}^{\sup}$ and $\mathbf{f}^{\inf}$. Here we can resort to Kirschenhofer, Prodinger, and Szpankowski [17, Theorem 1.2], where exact asymptotics of costs of partial match retrievals for fixed *periodic* queries are obtained. The method used there is a Mellin transform approach (cf. [9] for details) that, in our case, would make use of the representation

$$f_N^{\sup} = -\frac{1}{2\pi i} \int_{-1-i\infty}^{-1+i\infty} \frac{P(p^{-s}, q^{-s})(s+1)\Gamma(s)}{1 - p^{-s(K-u)}(p^{-s}+q^{-s})^u} N^{-s} ds \left(1 + \mathcal{O}\left(N^{-1}\right)\right),$$

where $P(\cdot,\cdot)$ is a polynomial of two variables of degree $K-1$, and which allows us, by invoking residue calculus, to derive

$$f_N^{\sup} = N^{\gamma_1} \xi(\ln N) + o\left(N^{\gamma_1}\right),$$

where ξ is a continuous positive periodic function which is constant if $\frac{\ln p}{\ln q}$ is irrational. For more details see also [26, Theorem 1, Lemma 4]. This proves the first part of (2), and the proof of the second part follows along the same lines, establishing $f_N^{\inf} = \Theta(N^{\gamma_2})$.

It remains to prove (3), which will be an immediate consequence of the following result.

LEMMA 4. *Assume that $(\rho_n)_{n\geq 1}$, as defined in* (13)*, is nicely approximated by a geometric sequence in the following sense. There is $r \in [q,p]$, an increasing sequence $(a_n)_{n\geq 1}$ of positive real numbers, and an increasing differentiable concave function g, defined on $[0,\infty[$, with $g(0)=0$, $a_n \leq g(n)$, and $g'(x) = o(1)$, as $x \to \infty$, such that*

$$|\ln(\rho_{Kn} r^{-(K-u)n}) - \ln(\rho_{Km} r^{-(K-u)m})| < a_n \vee g(|n-m|) \tag{16}$$

holds for $m,n \in \mathbb{N}$ with n sufficiently large. Then, denoting $\bar{S}_{\boldsymbol{\kappa},n} := \ln(\rho_{Kn} r^{-(K-u)n})$, we have

$$\ln f_{\boldsymbol{\kappa},N} = b \ln N + b\bar{S}_{\boldsymbol{\kappa},n_b} + \mathcal{O}\left(a_{n_b} + G(n_b) + \sqrt{\tfrac{g^3(n_b)}{n_b}} + \ln\ln N\right), \tag{17}$$

where b satisfies $(p^b+q^b)^u r^{(K-u)b} = 1$. The constant $c_b = b\big(u \ln(P_b^{-P_b} Q_b^{-Q_b})\big)^{-1}$ is as defined in Theorem 1, and $n_b = \lfloor c_b \ln N \rfloor$. Moreover $G(n) := \max_{x\geq 0}\big(bg(x) - d_b \frac{x^2}{2n}\big)$, with $d_b := \big(uc_b^2 P_b Q_b \ln^2 \frac{p}{q}\big)^{-1}$.

We are now ready to complete the proof of Theorem 1. First we claim the following: Equation (16) of Lemma 4 is satisfied for Q-almost every $\boldsymbol{\kappa} \in K_\omega$ if we choose $r = p^p q^q$, $g(x) = (\ln \frac{p}{q})\sqrt{2x\ln(x+1)}$, and $a_n = g(\sqrt{n})$.

Note that $\bar{S}_{\boldsymbol{\kappa},n}$ and $S_{\boldsymbol{\kappa},n}$, as defined in Theorem 1, coincide for $r = p^p q^p$, and that also $b = \beta$. We define the events

$$E_{n,m} := \{\boldsymbol{\kappa} \in K_\omega : |S_{\boldsymbol{\kappa},n} - S_{\boldsymbol{\kappa},m}| \geq a_n \vee g(|n-m|)\}.$$

Since $S_{\cdot,n} \overset{\mathcal{D}}{=} \ln \frac{p}{q} \sum_{i=1}^{n(K-u)} (Y_i - p)$, with $(Y_i)_{i\geq 1}$ i.i.d. and $\mathbb{P}(Y_1 = 1) = p = 1 - \mathbb{P}(Y_1 = 0)$, we obtain, applying Chernoff's bound (cf. [21]),

$$Q(E_{n,m}) \leq 2\exp\left(-2\frac{\sqrt{n}\ln n}{|n-m|}\right) \wedge 2\exp\left(-4\ln|n-m|\right).$$

Next we find $\sum_{m\geq 1} Q(E_{n,m}) = \mathcal{O}(n^{-3/2})$, and thus $\sum_{n,m\geq 1} Q(E_{n,m}) < \infty$. Now, by the Borel–Cantelli lemma, $Q(E_{n,m} \text{ i.o.}) = 0$, which proves the claim.

Next we are going to determine $G(n)$. We have to solve $bg'(x) = d_\beta \frac{x}{n}$ for x, which yields

$$x = x_n = \left(\frac{\beta^2 \ln^2 \frac{p}{q}}{12 d_\beta^2}\right)^{1/3} n^{2/3} (\ln n)^{1/3} \left(1 + \mathcal{O}\left(\frac{\ln\ln n}{\ln n}\right)\right).$$

So we obtain

$$G(n) = \beta g(x_n) - d_\beta \frac{x_n^2}{n} = \left(\frac{3\beta^4 \ln^4 \frac{p}{q}}{16 d_\beta}\right)^{1/3} n^{1/3} (\ln n)^{2/3} \left(1 + \mathcal{O}\left(\frac{\ln\ln n}{\ln n}\right)\right).$$

Thus, for n large enough, $a_n \vee G(n) \vee \sqrt{\frac{g^3(n)}{n}} = G(n) = \mathcal{O}(n^{1/3}(\ln n)^{2/3})$, which, via (17), proves (3), and thus completes the proof of Theorem 1. ∎

Remark 5. As a simple modification of the preceding proof shows, Lemma 4 allows us to generalize (3) and Corollary 1 to queries distributed according to a Bernoulli model that differs from the key model. Still we assume that the query is independent of the keys, but now a specified element of the query equals 0 with probability $\bar{p}$, and 1 with probability $\bar{q}$, which satisfy $r = p^{\bar{p}} q^{\bar{q}}$. Then (3) and Corollary 1 remain true if we replace $(S, \beta, c_\beta, \sigma)$ by $(\bar{S}, b, c_b, \bar{\sigma})$, with $\bar{\sigma} = b \ln \frac{p}{q} \sqrt{\left(\frac{K}{u} - 1\right)\bar{p}\bar{q} \big/ \ln\left(P_b^{-P_b} Q_b^{-Q_b}\right)}$.

Proof of Lemma 4. We only have to consider logarithms of $f_{\boldsymbol{\kappa},N}$, and thus the following crude estimate becomes useful:

$$f_{\boldsymbol{\kappa},N} = \Theta\left(\sum_{n\geq 0} \left[(\mathbf{M}_p + \mathbf{M}_q)^{un} \mathbf{M}_{\rho_{Kn}} \mathbf{e}\right]_N\right), \tag{18}$$

where the constants implied only depend on the dimension K. This follows from (11), from which we deduce, with the help of property 3 of Lemma 1,

$$\sum_{n\geq 0} \prod_{k=1}^{Kn} \mathbf{B}_k \mathbf{e} \preceq \mathbf{f}_{\boldsymbol{\kappa}} \preceq K \sum_{n\geq 0} \prod_{k=1}^{Kn} \mathbf{B}_k \mathbf{e}. \tag{19}$$

Recall that we put $P_a := \frac{p^a}{p^a + q^a}$ and $Q_a := 1 - P_a$ for $a \in \mathbb{R}$. We extend this definition by putting $P_{-\infty} := 0$ and $P_\infty := 1$, and we continue defining

$$k_n = k_n(N, \boldsymbol{\kappa}) := \log_{\frac{p}{q}}(N \rho_{Kn} p^{un}) \tag{20}$$

and

(21)
$$n_a = n_a(N, \boldsymbol{\kappa}) := \max(n \geq 0 : N\rho_{_{Kn}} p^{P_a un} q^{Q_a un} \geq 1) = \max(n \geq 0 : k_n \geq Q_a un).$$

By our assumption $p > q$, both P_a and n_a are increasing as a increases, and k_n is decreasing as n increases. Moreover $n_a = \Theta(\ln N)$ holds uniformly in $\boldsymbol{\kappa} \in K_\omega$ and $a \in [-\infty, \infty]$, since from $q^{(K-u)n} \leq \rho_{_{Kn}} \leq p^{(K-u)n}$ we can deduce $\lfloor \frac{1}{K} \log_{\frac{1}{q}} N \rfloor \leq n_a \leq \lfloor \frac{1}{K} \log_{\frac{1}{p}} N \rfloor$.

The first step towards (17) is the following result.

LEMMA 5. *Let $H : \mathbb{R}^2 \to \mathbb{R}$ be defined by*

$$H(n,k) := k \ln \frac{un}{k} + (un - k) \ln \frac{un}{un - k}. \tag{22}$$

Then, uniformly in $\boldsymbol{\kappa} \in K_\omega$,

$$\ln f_{\boldsymbol{\kappa},N} = \max_{n_0 \leq n \leq n_2} H(n, k_n) + \mathcal{O}(\ln \ln N). \tag{23}$$

Proof. We observe that $E_N(x) = \Theta(1 \wedge N^2 x^2)$, with $E_N(x)$ defined in (15); thus by (18)

$$f_{\boldsymbol{\kappa},N} \asymp \sum_{n \geq 0} \sum_{k=0}^{un} \binom{un}{k} E_N\left(\rho_{_{Kn}} p^{un-k} q^k\right) \asymp \sum_{n \geq 0} \underbrace{\sum_{k=0}^{un} \binom{un}{k} \left(1 \wedge \left(N\rho_{_{Kn}} p^{un-k} q^k\right)^2\right)}_{\Sigma_{N,n}}.$$

Note that $\Sigma_{N,n}$ grows (resp., decays) like a geometric sequence for $n \leq n_0$ (resp., $n > n_2$); more precisely, we have

$$\Sigma_{N,n} \leq 2^{un} \wedge N^2 \rho_{_{Kn}}^2 (p^2 + q^2)^{un}$$

for any $n \geq 0$, and moreover $\Sigma_{N,n} \geq \sum_{k \leq Q_0 un} \binom{un}{k} \geq 2^{un-1}$ for $n \leq n_0$, and

$$\begin{aligned} \Sigma_{N,n} &\geq \sum_{k \geq Q_2 un} \binom{un}{k} \left(N\rho_{_{Kn}} p^{un-k} q^k\right)^2 = N^2 \rho_{_{Kn}}^2 (p^2 + q^2)^{un} \sum_{k \geq Q_2 un} \binom{un}{k} P_2^{un-k} Q_2^k \\ &\geq c N^2 \rho_{_{Kn}}^2 (p^2 + q^2)^{un}, \end{aligned}$$

for $n > n_2$ and some $c > 0$. (It follows from results in [13] that we can choose $c = \frac{1}{4} \wedge Q_2$.) Thus $\sum_{n=0}^{n_0} \Sigma_{N,n} = \mathcal{O}(\Sigma_{N,n_0})$ and $\sum_{n \geq n_2} \Sigma_{N,n} = \mathcal{O}(\Sigma_{N,n_2})$. For $n_0 \leq n \leq n_2$ we have for any $a \in [0, 2]$

$$\Sigma_{N,n} \leq \sum_{k=0}^{un} \binom{un}{k} \left(N\rho_{_{Kn}} p^{un-k} q^k\right)^a = (N\rho_{_{Kn}})^a (p^a + q^a)^{un}, \tag{24}$$

and a_n, which minimizes the right-hand side of (24), obeys

$$\ln(N\rho_{_{Kn}}) + un(P_{a_n} \ln p + Q_{a_n} \ln q) = 0.$$

Note that $n_{a_m} = m$; cf. (21). This results in

$$\Sigma_{N,n} \leq \left(P_{a_n}^{-P_{a_n}} Q_{a_n}^{-Q_{a_n}}\right)^{un} = e^{H(n,k_n)},$$

where we used $Q_{a_n} un = \log_{\frac{p}{q}}(N\rho_{K_n} p^{un}) = k_n$. Furthermore, $Q_2 un_0 \le k_n \le Q_0 un_2$ implies $k_n = \Theta(\ln N)$, uniformly in $\boldsymbol{\kappa} \in K_\omega$ and $n_0 \le n \le n_2$. This is used, together with Stirling's formula, in the following lower estimate:

$$\Sigma_{N,n} \ge \binom{un}{\lfloor k_n \rfloor} = \Omega\left(\frac{e^{H(n,k_n)}}{\sqrt{\ln N}}\right).$$

Thus, $(\max_{n_0\le n\le n_2} e^{H(n,k_n)})^{-1} \sum_{n=n_0}^{n_2} \Sigma_{N,n}$ is upper bounded by $n_2 - n_0 + 1$ and lower bounded by $\Omega(\ln^{-\frac{1}{2}} N)$, so we finally obtain

$$\ln f_{\boldsymbol{\kappa},N} = \ln\left(\sum_{n=n_0}^{n_2} \Sigma_{N,n}\right) + \mathcal{O}(1) = \max_{n_0\le n\le n_2} H(n,k_n) + \mathcal{O}(\ln\ln N), \tag{25}$$

which completes the proof of the lemma. □

Some properties of the function H are explored in the following three lemmas.

LEMMA 6. *If either $n_0 \le m \le n \le n_{\gamma_2}$ or $n_2 \ge m \ge n \ge n_{\gamma_1}$, then*

$$H(n,k_n) \ge H(m,k_m). \tag{26}$$

Proof. First observe that $-\log_{\frac{p}{q}} p^{K-u}p^u \le k_m - k_{m+1} \le -\log_{\frac{p}{q}} q^{K-u}p^u$. We let

$$h(t) := H(m+t, k_m + (k_{m+1}-k_m)t)$$

and assume $n_0 \le m < n_{\gamma_2}$, which implies $a_{m+1} \le \gamma_2$. Next we apply the mean value theorem. For some $0 < \tau < 1$ and $a_m < \bar a < a_{m+1}$ we have

$$\begin{aligned}
H(m+1,k_{m+1}) - H(m,k_m) &= h(1) - h(0) = h'(\tau)\\
&= u\ln\frac{u(m+\tau)}{u(m+\tau) - k_m - (k_{m+1}-k_m)\tau}\\
&\quad + (k_{m+1}-k_m)\ln\frac{u(m+\tau) - k_m - (k_{m+1}-k_m)\tau}{k_m + (k_{m+1}-k_m)\tau}\\
&= u\ln\frac{1}{P_{\bar a}} + (k_{m+1}-k_m)\ln\frac{P_{\bar a}}{Q_{\bar a}}\\
&\ge u\ln\frac{1}{P_{\bar a}} + \log_{\frac{p}{q}}(q^{K-u}p^u)\ln\frac{P_{\bar a}}{Q_{\bar a}} = u\ln\frac{1}{P_{\bar a}} + \bar a\ln q^{K-u}p^u\\
&= \ln\left((p^{\bar a}+q^{\bar a})^u q^{(K-u)\bar a}\right) \ge \ln\left((p^{\gamma_2}+q^{\gamma_2})^u q^{(K-u)\gamma_2}\right) = 0.
\end{aligned}$$

This proves (26) in the case $n_0 \le m \le n \le n_{\gamma_2}$. The proof in the case $n_2 \ge m \ge n \ge n_{\gamma_1}$ follows similar lines. □

LEMMA 7. *For $0 \le b \le 1$ and $Q_1 un \le k, k' \le Q_0 un$ we have*

$$\left|H(n,k) - H(n,k') - (k-k')b\ln\frac{p}{q}\right| \le \frac{2|k-k'|}{qun}\left(|k - Q_b un| \vee |k' - Q_b un|\right). \tag{27}$$

Proof. Assume without loss of generality (w.l.o.g.) that $k \ge k'$. The left-hand side of (27) equals $\left|\int_{k'}^{k} \ln\frac{P_b x}{Q_b(un-x)}dx\right|$ and is less than $(k-k')\max_{k'\le x\le k}\left|\ln\frac{P_b x}{Q_b(un-x)}\right|$. The latter absolute value can be estimated as follows:

$$\left|\ln\frac{P_b x}{Q_b(un-x)}\right| = \left|\int_{Q_b(un-x)}^{P_b x}\frac{d\xi}{\xi}\right| \le \frac{|Q_b un - x|}{un}\left(\frac{1}{P_bQ_1} \vee \frac{1}{Q_bP_0}\right).$$

Furthermore $P_bQ_1 \wedge Q_bP_0 \geq Q_1P_0 = \frac{q}{2}$. □

LEMMA 8. *Let $r \in [q,p]$ and $b \in \mathbb{R}$ be such that $r^{(K-u)b}(p^b+q^b)^u = 1$. Moreover let k_n and n_a be defined by (20) and (21), with $\rho_{Kn} = r^{(K-u)n}$, and for $a \in [-\infty,\infty]$ let ν_a be such that $N\left(r^{K-u}p^{uP_a}q^{uQ_a}\right)^{\nu_a} = 1$.*

Then $\phi_N(n) := H(n,k_n)$ is a concave function, defined for real $n \in]\nu_{-\infty},\nu_\infty[$, which satisfies

$$\phi_N(n) = b\ln N - d_b\frac{(n-n_b)^2}{2n_b} + \mathcal{O}\left(\frac{1+|n-n_b|}{n_b} + \frac{|n-n_b|^3}{n_b^2}\right), \tag{28}$$

and for some $C > 0$

$$\phi_N(n) - b\ln N \leq -C\frac{(n-\nu_b)^2}{2\nu_b}. \tag{29}$$

Proof. First note that $n_a = \lfloor \nu_a \rfloor$ for $-\infty \leq a \leq \infty$, and that

$$\nu_b = \frac{\ln N}{\ln r^{-(K-u)}p^{-P_b}q^{-Q_b}} = c_b\ln N,$$
$$\nu_{-\infty} = \frac{\ln N}{\ln r^{-(K-u)}q^{-u}}, \quad \nu_\infty = \frac{\ln N}{\ln r^{-(K-u)}p^{-u}}.$$

We observe that $\frac{k_n}{un}$ decreases from 1 to 0, as n increases from $\nu_{-\infty}$ to ν_∞; more precisely

$$\frac{k_n}{un} = \frac{1}{un}\log_{\frac{p}{q}} N + \frac{1}{u}\log_{\frac{p}{q}} r^{(K-u)}p^u = \frac{\nu_{-\infty}}{\nu_{-\infty}-\nu_\infty}\left(1-\frac{\nu_\infty}{n}\right),$$

and thus ϕ_N is well defined on $]\nu_{-\infty},\nu_\infty[$. We are seeking the Taylor expansion of ϕ_N at its maximum, and we derive, along lines similar to the proof of Lemma 6,

$$\begin{aligned}\phi_N'(\nu_a) &= u\ln\frac{u\nu_a}{u\nu_a - k_{\nu_a}} + \ln\frac{u\nu_a - k_{\nu_a}}{k_{\nu_a}}\left.\frac{dk_n}{dn}\right|_{n=\nu_a} \\ &= u\ln\frac{1}{P_a} + \log_{\frac{p}{q}}(r^{K-u}p^u)\ln\frac{P_a}{Q_a} = \ln\left((p^a+q^a)^u r^{(K-u)a}\right),\end{aligned}$$

which equals 0 iff $a = b$. Next

$$\phi_N(\nu_b) = u\nu_b\left(P_b\ln\frac{1}{P_b} + Q_b\ln\frac{1}{Q_b}\right) = \frac{b}{c_b}\nu_b = b\ln N.$$

Furthermore

$$\phi_N''(n) = \frac{u}{n} - \frac{u\nu_\infty}{\nu_\infty-\nu_{-\infty}}\frac{1}{n-\nu_{-\infty}} - \frac{u\nu_{-\infty}}{\nu_\infty-\nu_{-\infty}}\frac{1}{\nu_\infty-n} = -\frac{u\nu_{-\infty}\nu_\infty}{n(n-\nu_{-\infty})(\nu_\infty-n)}$$

is easily seen to be strictly negative for $n \in]\nu_{-\infty},\nu_\infty[$, which proves that ϕ_N is

concave on that interval. Now we compute

$$\begin{aligned}
\phi_N''(\nu_b) &= -\frac{u}{\nu_b}\left(\frac{\nu_b}{\nu_{-\infty}}-1\right)^{-1}\left(1-\frac{\nu_b}{\nu_\infty}\right)^{-1}\\
&= -\frac{u}{c_b^2\nu_b}\left(\ln r^{-(K-u)}q^{-u}-\frac{1}{c_b}\right)^{-1}\left(\frac{1}{c_b}-\ln r^{-(K-u)}p^{-u}\right)^{-1}\\
&= -\frac{b^2}{uc_b^2\nu_b}\left(\ln\frac{1}{Q_b}-\ln P_b^{-P_b}Q_b^{-Q_b}\right)^{-1}\left(\ln P_b^{-P_b}Q_b^{-Q_b}-\ln\frac{1}{P_b}\right)^{-1}\\
&= -\frac{1}{uc_b^2\nu_b}\left(P_bQ_b\ln^2\frac{p}{q}\right)^{-1}\\
&= -\frac{d_b}{\nu_b};
\end{aligned}$$

moreover, $\int_0^{n-\nu_b}\frac{x^2}{2!}\phi_N'''(n-x)dx$, the remainder in the order two Taylor expansion of ϕ_N at ν_b, can be shown to be $\mathcal{O}\big(\frac{|n-\nu_b|^3}{\nu_b^2}\big)$ as $N\to\infty$ and uniformly for $\nu_{-\infty}<n<\nu_\infty$, using $\nu_\infty-\nu_b=\Theta(\nu_b)$ and $\nu_b-\nu_{-\infty}=\Theta(\nu_b)$. We thus derive

$$\phi_N(n)=b\ln N-d_b\frac{(n-\nu_b)^2}{2\nu_b}+\mathcal{O}\left(\frac{|n-\nu_b|^3}{\nu_b^2}\right).$$

Now, replacing ν_b by n_b yields (28), with the slightly changed error term. The following upper estimate for ϕ_N'' on the interval $]\nu_{-\infty},\nu_\infty[$ yields the definition of the constant C, needed for the proof of (29):

$$\phi_N''(n)\le-\frac{u\nu_{-\infty}}{(n-\nu_{-\infty})(\nu_\infty-n)}\le-\frac{4u\nu_{-\infty}}{(\nu_\infty-\nu_{-\infty})^2}=:-\frac{C}{\nu_b}.\qquad\square$$

We proceed in the proof of Lemma 4. Our aim is to approximate in (23) the terms $H(n,k_n)$ corresponding to $\rho_{_{Kn}}$ by terms $H(n,k_n')$ corresponding to the approximating sequence $r^{(K-u)n}$. Note that k_n is defined in (20), and we let

$$k_n':=\log_{\frac{p}{q}}Nr^{(K-u)n}p^{un}.$$

Putting $m=0$ in (16) yields

$$|\ln(\rho_{_{Kn}}r^{-(K-u)n})|=|k_n-k_n'|\ln\frac{p}{q}=o(n).$$

Thus $k_n\in[Q_{\gamma_1}un,Q_{\gamma_2}un]$ implies $k_n'\in[Q_1un,Q_0un]$ for large enough n, and we can deduce

$$|H(n,k_n)-H(n,k_n')|=\mathcal{O}(|k_n-k_n'|)=\mathcal{O}\left(g(\nu_b)+g(|n-\nu_b|)\right),\tag{30}$$

where the left equality follows easily from Lemma 7, and the right equality is implied by

$$\begin{aligned}
|k_n-k_n'|\ln\frac{p}{q}&\le|k_{n_b}-k_{n_b}'|\ln\frac{p}{q}+|k_n-k_n'-(k_{n_b}-k_{n_b}')|\ln\frac{p}{q}\\
&=|\ln(\rho_{_{Kn_b}}r^{-(K-u)n_b})|+|\ln(\rho_{_{Kn}}r^{-(K-u)n})-\ln(\rho_{_{Kn_b}}r^{-(K-u)n_b})|\\
&=\mathcal{O}\left(g(\nu_b)+g(|n-\nu_b|)\right),
\end{aligned}$$

which holds by the assumptions made in (16).

Lemma 6 and (29) and (30) will be used in the following, where we show that the max in (23) is attained on a small subset of $[n_0, n_2]$:

$$\begin{aligned}\max_{n_0\le n\le n_2} H(n,k_n) &= \max_{n_{\gamma_2}\le n\le n_{\gamma_1}} H(n,k_n)\\ &= \max_{n_{\gamma_2}\le n\le n_{\gamma_1}} (H(n,k_n) - H(n,k'_n) + H(n,k'_n))\\ &\le b\ln N + \mathcal{O}(g(\nu_b)) + \max_{n_{\gamma_2}\le n\le n_{\gamma_1}} \left(\mathcal{O}(g(|n-\nu_b|)) - C\frac{(n-\nu_b)^2}{2\nu_b}\right).\end{aligned}$$

Defining $\operatorname{argmax}\{F(x), x\ge 0\} := \sup\left\{x\ge 0 : F(x) = \sup_{y\ge 0} F(y)\right\}$ for a real function F, we obtain

$$\begin{aligned}x^* :&= \operatorname{argmax}\left\{\mathcal{O}(g(x)) - C\frac{x^2}{2\nu_b}, x\ge 0\right\}\\ &\le \sup\left\{x\ge 0 : \mathcal{O}(g(x)) - C\frac{x^2}{2\nu_b}\ge 0\right\}\\ &\le \sup\left\{x\ge 0 : \mathcal{O}(g(\nu_b)) - C\frac{x^2}{2\nu_b}\ge 0\right\}\\ &= \mathcal{O}\left(\sqrt{\nu_b g(\nu_b)}\right),\end{aligned}$$

where the first sup yields the estimate $x^* = \mathcal{O}(\nu_b)$, which, together with $g(\mathcal{O}(n)) = \mathcal{O}(g(n))$, is then used to obtain the second inequality. Of course, the constants implied by $\mathcal{O}$ can vary from line to line, but as they are derived from Lemma 7, they are independent of $\boldsymbol{\kappa}$. The maximizing n has to belong to the set

$$\mathcal{N}_b := \left\{n\in\mathbb{N} : |n-\nu_b| = \mathcal{O}\left(\sqrt{\nu_b g(\nu_b)}\right)\right\}.$$

We continue observing that

$$|k_n - Q_b u n| \vee |k'_n - Q_b u n| = \mathcal{O}\left(g(\nu_b) + |n-\nu_b|\right),$$

which allows us to rewrite the error term of Lemma 7:

$$\begin{aligned}&\frac{2|k_n-k'_n|}{qun_b}\left(|k_n - Q_b u n| \vee |k'_n - Q_b u n|\right)\\ &\qquad = \mathcal{O}\left(\frac{g^2(\nu_b)}{n_b} + \frac{g(\nu_b)|n-\nu_b|}{n_b} + \frac{g(|n-\nu_b|)|n-\nu_b|}{n_b}\right).\end{aligned}$$

Now we can be more precise, again employing Lemmas 7 and 8:

(31)
$$\begin{aligned}&\max_{n_0\le n\le n_2} H(n,k_n) = \max_{n\in\mathcal{N}_b}\left(H(n,k_n) - H(n,k'_n) + H(n,k'_n)\right)\\ &\quad= \max_{n\in\mathcal{N}_b}\left[(k_n-k'_n)b\ln\frac{p}{q} + \mathcal{O}\left(\frac{|k_n-k'_n|}{n_b}\left(|k_n - Q_b u n| \vee |k'_n - Q_b u n|\right)\right)\right.\\ &\qquad\qquad \left. + b\ln N - d_b\frac{(n-n_b)^2}{2n_b} + \mathcal{O}\left(\frac{1+|n-n_b|}{n_b} + \frac{|n-n_b|^3}{n_b^2}\right)\right]\\ &\quad= b\ln N + b\bar{S}_{\boldsymbol{\kappa},n_b} + \mathcal{O}\left(\sqrt{\tfrac{g^3(n_b)}{n_b}}\right) + \max_{n\in\mathcal{N}_b}\left[-d_b\frac{(n-n_b)^2}{2n_b} + b(\bar{S}_{\boldsymbol{\kappa},n} - \bar{S}_{\boldsymbol{\kappa},n_b})\right].\end{aligned}$$

By the definition of G we have

$$
\begin{aligned}
0 &\le \max_{n\in\mathcal{N}_b}\left[-d_b\frac{(n-n_b)^2}{2n_b}+b(\bar{S}_{\boldsymbol{\kappa},n}-\bar{S}_{\boldsymbol{\kappa},n_b})\right]\\
&\le \max_{n\in\mathcal{N}_b}\left[-d_b\frac{(n-n_b)^2}{2n_b}+b\big(a_{n_b}\vee g(|n-n_b|)\big)\right]\\
&\le a_{n_b}\vee G(n_b).
\end{aligned}
\tag{32}
$$

Now, fitting Lemma 5 and equations (31) and (32) together finishes the proof of Lemma 4. □

Remark 6. It is clear that we need not be that precise about the constant d_b in Lemma 4. Since the function G satisfies $G(n)\ge G(cn)\ge cG(n)$ for $0<c\le 1$, i.e., $G(cn)=\Theta(G(n))$ for fixed $c>0$ (actually G can be shown to be increasing, concave, and satisfying $\lim_{x\searrow 0}G(x)=0$), it is only $d_b>0$ which matters. There are, however, two instances where we could wish to have the precise value. To these we devote the following remarks.

Remark 7. Suppose we want to get hold of what is hidden behind the error terms in (17), when $\boldsymbol{\kappa}$ is distributed according to Q, in which case we have $b=\beta$. Reworking the preceding proof with $g(x)=(\ln\frac{p}{q})\sqrt{2x\ln(x+1)}$, we find that the terms estimated by $\mathcal{O}\big(\sqrt{g^3(n_\beta)/n_\beta}\,\big)$ are indeed of order $\mathcal{O}(n_\beta^{1/6}\ln^{5/6}n_\beta)$. Furthermore, $a_{n_\beta}+G(n_\beta)$ is an upper bound for the quantity

$$
Y_{n_\beta}:=\max_{n\ge 0}\left(\beta(S_n-S_{n_\beta})-d_\beta\frac{(n-n_\beta)^2}{2n_\beta}\right).
$$

Donsker's invariance principle (and scaling properties of Brownian motion) can be used to show that $D_\beta^{-1}(c_\beta/\beta)^{1/3}n_\beta^{-1/3}Y_{n_\beta}$ converges in distribution to the random variable

$$
Y:=\sup_{t\in\mathbb{R}}(W_t-t^2),
$$

where W is a two-sided Brownian motion, and

$$
D_\beta:=\frac{\beta^2\ln^2\frac{p}{q}}{\ln\left(P_\beta^{-P_\beta}Q_\beta^{-Q_\beta}\right)}\left(2p^2q^2\left(\frac{K}{u}-1\right)^2P_\beta Q_\beta\right)^{1/3}.
$$

Komlós–Major–Tusnády-type strong approximation results (cf. [6]) then even allow us to deduce

$$
\mathbb{E}\,\ln f_{\boldsymbol{\kappa},N}=\beta\ln N+D_\beta\mathbb{E}\,Y(\beta\ln N)^{1/3}+\mathcal{O}\left(\ln^{1/6}N\,\ln^{5/6}\ln N\right).
$$

Moreover, since $Q\big(\beta(S_{\boldsymbol{\kappa},n^3+n^2}-S_{\boldsymbol{\kappa},n^3})\ge\big(\frac{d_\beta}{2}+2D_\beta\big(\frac{\beta}{c_\beta}\big)^{1/3}\mathbb{E}\,Y\big)n,\ \text{i.o.}\big)=1$, by the second Borel–Cantelli lemma, we easily deduce

$$
Q\left(\ln f_{\boldsymbol{\kappa},N}-\beta\ln N-\beta S_{\boldsymbol{\kappa},\lfloor c_\beta\ln N\rfloor}\ge D_\beta\mathbb{E}\,Y(\beta\ln N)^{1/3},\ \text{i.o.}\right)=1.
$$

Since in the example depicted in Figure 2 the constant $D_\beta\mathbb{E}\,Y\approx 0.02$ is very small, we do not observe the gaps of size $\Theta(\ln^{1/3}N)$ between the two curves in the plotted range.

The distribution of Y surfaces in a paper of Chernoff [5] and was studied analytically by Groeneboom [12]. There also seems to be a close connection to results obtained by Steinsaltz in his thesis [29], where he develops general methods to investigate for a certain class of processes the difference of expected maximum and maximum expectation, which in many cases turns out to be approximately distributed like a third root term times the random variable Y.

Remark 8. It is of course essential to know the dependence on p of the error terms in (17) when we want to investigate the phase change from standard normal to Dirac at 0 of $\frac{C_N - \mathbb{E}\, C_N}{\sqrt{\operatorname{Var} C_N}}$ at the transition from the symmetric to the asymmetric Bernoulli models. One would have to vary p with N, and $p = p_N = \frac{1}{2} + \lambda/\sqrt{\ln N}$ with $\lambda \geq 0$ seems to be the right choice.

Remark 9. A heuristic derivation of the values of b and n_b appearing in Lemma 4 could be along the following lines. Suppose that $\boldsymbol{\kappa}$ satisfies $\ln(\rho_{Kn} r^{-(K-u)n}) = \mathcal{O}(1)$, which is a stronger regularity assumption than (16). With μ_n defined in (12), we let K_n denote the set of the 2^{μ_n} prefixes of length n that match $\boldsymbol{\kappa}$. Given N keys, the number of those keys having the prefix $\pi \in K_n$ is denoted N_π, and it is binomially distributed. Suppose now that for some $c > 0$ we have $f^i_{\boldsymbol{\kappa},N} = \Theta(N^c)$, uniformly in i. Then (10) is iterated $n-1$ times, and computing expectations, we obtain $f_{\boldsymbol{\kappa},N} = \mathcal{O}(2^{\mu_n}) + \mathbb{E} \sum_{\pi \in K_n} f^n_{\boldsymbol{\kappa},N_\pi}$, and thus $\Theta(N^c) = \mathcal{O}(2^{\mu_n}) + \Theta\left(\sum \binom{\mu_n}{k} (N \rho_n p^k q^{\mu_n - k})^c\right)$. Since the latter has to hold for $N, n \to \infty$ in such a way that $n = o(\ln N)$, we must have $c = b$. Suppose further that $F^i_{\boldsymbol{\kappa},N} / f^i_{\boldsymbol{\kappa},N} \to 1$ in probability as $N \to \infty$, uniformly in i. Then, for large N, keys with prefix π (whose $*$-positions contain k zeros and $\mu_n - k$ ones) will contribute a proportion of

$$\mathbb{E}\, F^n_{\boldsymbol{\kappa},N_\pi} / f_{\boldsymbol{\kappa},N} = \Theta\left((\rho_n p^k q^{\mu_n - k})^b\right) = \Theta(P_b^k Q_b^{\mu_n - k})$$

to the total cost with high probability. This suggests that we should consider a different probabilistic model, where keys are distributed according to the Bernoulli(P_b) model, but conditioned to match $\boldsymbol{\kappa}$. Denoting by $\tilde{F}_{\boldsymbol{\kappa},N}$ (resp., $\tilde{I}_N$) the cost of a partial match query for $\boldsymbol{\kappa}$ (resp., the number of internal nodes) in the new model, we have $\tilde{F}_{\boldsymbol{\kappa},N} = \Theta(\tilde{I}_N)$ and $\mathbb{E}\, \tilde{I}_N = \Theta(N)$, and therefore also

$$\mathbb{E}\, \tilde{F}^n_{\boldsymbol{\kappa},N_\pi} / \tilde{f}_{\boldsymbol{\kappa},N} = \Theta(P_b^k Q_b^{\mu_n - k}),$$

as $N \to \infty$, with high probability, where $\tilde{f}_{\boldsymbol{\kappa},N} := \mathbb{E}\, \tilde{F}_{\boldsymbol{\kappa},N}$. This might convince us that there is an asymptotic equivalence between the levels with highest expected contribution to $f_{\boldsymbol{\kappa},N}$ and $\tilde{f}_{\boldsymbol{\kappa},\lfloor N^b \rfloor}$. Now any level ν, which contributes most to $\mathbb{E}\, \tilde{I}_N$, satisfies $\nu \sim \frac{\ln N}{h}$, where $h = -P_b \ln P_b - Q_b \ln Q_b$ denotes the entropy of the Bernoulli(P_b) distribution. This follows from the $u = K$ case of Lemma 8, but one could also argue that only with that choice of ν do typical prefixes of length ν have probabilities of order $N^{-1+o(1)}$, or one could use the fact that the depth D_N of a full external node chosen uniformly at random in a trie built from N keys according to the Bernoulli(P_b) model is concentrated around its expected value $\mathbb{E}\, D_N \sim \frac{\ln N}{h}$. See Szpankowski's book [30] for the *asymptotic equipartition property* and other entropy related issues, as well as for references and results on depth in digital data structures. From $\mu_n \sim Kn/u$ we deduce that the main contribution to $\tilde{f}_{\boldsymbol{\kappa},\lfloor N^b \rfloor}$ comes from levels $\sim \frac{Kb \ln N}{uh}$, and the value of n_b becomes clear, observing that Lemma 4 is stated in terms of Kn rather than n. Finally one has to argue that n_b is insensitive to small irregularities of the query, as allowed by (16), but that due to those irregularities, the expected contributions to $f_{\boldsymbol{\kappa},N}$ from levels $n \sim n_b$ change by a factor of $(\rho_{Kn} r^{-(K-u)n})^b = e^{b \bar{S}_{\boldsymbol{\kappa},n}}$.

4. The variance. Note that $\boldsymbol{\kappa}$ is fixed throughout this section, and dependencies on $\boldsymbol{\kappa}$ will mostly be suppressed.

Proof of Theorem 2. Subtracting (10) from (6), squaring, and taking expectations, we obtain

$$\mathbf{v}^i = \mathbf{s}^i + \mathbf{B}_{i+1}\mathbf{v}^{i+1}, \quad i \geq 0, \tag{33}$$

where

$$s_N^i = \sum_{k=0}^{N} \binom{N}{k} p^k q^{N-k} \left(a_{0,i+1}(f_k^{i+1} - (\mathbf{M}_p \mathbf{f}^{i+1})_N) + a_{1,i+1}(f_{N-k}^{i+1} - (\mathbf{M}_q \mathbf{f}^{i+1})_N)\right)^2. \tag{34}$$

Iterating, we find

$$\mathbf{v} = \mathbf{I}\mathbf{s}^0 + \mathbf{B}_1\mathbf{s}^1 + \mathbf{B}_1\mathbf{B}_2\mathbf{s}^2 + \cdots = \sum_{n\geq 0}\prod_{k=1}^{n} \mathbf{B}_k \mathbf{s}^n. \tag{35}$$

The $\mathcal{O}$ result will follow from the following more general result, which holds for fixed nonnegative integers m, uniformly in $\boldsymbol{\kappa} \in K_\omega$:

$$\Delta^m v_N = \mathcal{O}\left(\frac{f_N}{N^m} \ln N\right). \tag{36}$$

For its proof we need more information about the sequences $\mathbf{s}^i$ and thus about the sequences $\Delta^m \mathbf{f}^i$ for $m \geq 0$.

LEMMA 9. *For any fixed integer $m \geq 0$ we have*

$$\Delta^m f_N^i = \mathcal{O}\left(\frac{f_N^i}{N^m}\right), \qquad \Delta^m s_N^i = \mathcal{O}\left(\frac{(f_N^i)^2}{N^{m+1}}\right), \tag{37}$$

as $N \to \infty$, where the constants implied by the $\mathcal{O}$ symbols depend on neither i nor $\boldsymbol{\kappa} \in K_\omega$.

Proof. Throughout the proof i is fixed. Since $|\Delta^m \mathbf{e}_N| \leq (m+1)! \frac{e_{N+m}}{(N+1)\cdots(N+m)}$, we can apply property 4 of Lemma 1 to the representation (14). Furthermore we use $f_{N+m}^i = \mathcal{O}(f_N^i)$, which we deduce from item 1 of Lemma 2. This proves the first statement regarding differences of $\mathbf{f}^i$, which is now used, together with item 1 of Lemma 2, to estimate the remainder term in the following Newton-type expansion of f_k^i around $k = \lfloor pN \rfloor$:

$$f_k^i = \sum_{\mu=0}^{M} \Delta^\mu f_{\lfloor pN \rfloor}^i \frac{(k - \lfloor pN \rfloor)^{\underline{\mu}}}{\mu!} + \mathcal{O}\left(\frac{f_N^i}{N^{M+1}} |k - pN|^{M+1}\right). \tag{38}$$

We fix N and obtain a similar expansion of $f_{N'-k}^i$ around $k = \lfloor pN \rfloor + \nu$, where $N' = N + \nu$ and $\nu = \mathcal{O}(1)$,

$$f_{N'-k}^i = \sum_{\mu=0}^{M} \Delta^\mu f_{\lceil qN \rceil}^i \frac{(\lfloor pN \rfloor - k + \nu)^{\underline{\mu}}}{\mu!} + \mathcal{O}\left(\frac{f_N^i}{N^{M+1}} (|k - pN|^{M+1} + 1)\right),$$

and, expanding $(k - \lfloor pN \rfloor)^{\underline{\mu}}$ and $(\lfloor pN \rfloor - k + \nu)^{\underline{\mu}}$ in powers of $k - pN'$, we obtain

$$a_{0,i} f_k^i + a_{1,i} f_{N'-k}^i = f_N^i \sum_{\mu=0}^{M} N^{-\mu} P_\mu(k - pN', \nu) + \mathcal{O}\left(\frac{f_N^i}{N^{M+1}} (|k - pN|^{M+1} + 1)\right),$$

where P_μ is a polynomial of two variables of degree μ, which depends on i and N. More precisely, the coefficients of P_μ involve the quantities $\{pN\} := pN - \lfloor pN \rfloor$, $\frac{N^\mu}{f_N^i}\Delta^\mu f^i_{\lfloor pN \rfloor}$, and $\frac{N^\mu}{f_N^i}\Delta^\mu f^i_{\lceil qN \rceil}$ but are uniformly bounded in i and N. Notice that in particular $P_0 = a_{0,i} f^i_{\lfloor pN \rfloor}/f^i_N + a_{1,i} f^i_{\lceil qN \rceil}/f^i_N$ and

$$P_1 = a_{0,i}\frac{N}{f_N^i}\Delta f^i_{\lfloor pN \rfloor}(k - pN' + \nu p + \{pN\}) + a_{1,i}\frac{N}{f_N^i}\Delta f^i_{\lceil qN \rceil}(pN' - k + \nu q - \{pN\}).$$

For k following a binomial distribution $B(N', p)$, we now have

$$\begin{aligned} s^{i-1}_{N'} &= \operatorname{Var}\left(a_{0,i} f^i_k + a_{1,i} f^i_{N'-k}\right) \\ &= (f^i_N)^2 \sum_{\mu,\mu'=1}^{M} N^{-\mu-\mu'} \operatorname{Cov}\left(P_\mu\left(k - pN', \nu\right), P_{\mu'}\left(k - pN', \nu\right)\right) \\ &\quad + \mathcal{O}\left((f^i_N)^2 N^{-\frac{M}{2}-1}\right). \end{aligned} \tag{39}$$

Note that we can start the summation with $\mu, \mu' = 1$, since $\operatorname{Cov}(P_\mu, P_{\mu'}) = 0$ if $\mu \wedge \mu' = 0$. For $\mu \wedge \mu' \geq 1$ we find

$$\operatorname{Cov}(P_\mu, P_{\mu'}) = \sum_{j=1}^{\lfloor \frac{\mu+\mu'}{2} \rfloor} \bar{P}_j(N') p_{\mu+\mu'-2j}(\nu),$$

where $\bar{P}_j$ and p_j are polynomials of degree j, with coefficients uniformly bounded in i and N; thus, computing differences with respect to ν, we obtain

$$\begin{aligned} \Delta^m \operatorname{Cov}(P_\mu, P_{\mu'}) &= \Delta^m \sum_{j=1}^{\lfloor \frac{\mu+\mu'}{2} \rfloor} \bar{P}_j(N+\nu) p_{\mu+\mu'-2j}(\nu) \\ &= \begin{cases} \mathcal{O}\left(N^{\lfloor \frac{\mu+\mu'-m}{2} \rfloor}\right) & \text{for } m < \mu + \mu', \\ 0 & \text{for } m \geq \mu + \mu'; \end{cases} \end{aligned}$$

thus the second statement of the lemma follows with $M = 2m$ from computing the mth differences in (39) and from item 2 of Lemma 2. The special case $m = 1$ of (39) reads

$$s^i_N = Npq\left(a_{0,i+1}\Delta f^{i+1}_{\lfloor pN \rfloor} - a_{1,i+1}\Delta f^{i+1}_{\lceil qN \rceil}\right)^2 + \mathcal{O}\left(\frac{(f^i_N)^2}{N^2}\right). \qquad \square$$

LEMMA 10. *The sequences $\mathbf{x}^i$ and $\mathbf{y}^i$, defined for $i \geq 0$ and some integer $m \geq 0$ by $x^i_N = N^{\underline{m}}\Delta^m s^i_{N-m}$ and $y^i_N = N^{\underline{m}}\Delta^m v^i_{N-m}$, satisfy the system of equations*

$$\mathbf{y}^i = \mathbf{x}^i + \mathbf{B}_{i+1}\mathbf{y}^{i+1}, \quad i \geq 0. \tag{40}$$

If in (40) the sequences $\mathbf{x}^i$ satisfy $|x^i_N| \leq c \cdot f^i_N$ for some $c > 0$ and $i, N \geq 0$, then $\mathbf{y}^0$ satisfies $y^0_N = \mathcal{O}(f_N \ln N)$, and the $\mathcal{O}$-constant is uniform in $\boldsymbol{\kappa} \in K_\omega$.

Proof. We multiply both sides of (33) with the matrix $\mathbf{D}_m$, which is defined by

$$(\mathbf{D}_m)_{N,k} = N^{\underline{m}}\binom{m}{N-k}(-1)^{N-k},$$

and obtain (40), since $\mathbf{x}^i = \mathbf{D}_m \mathbf{s}^i$, $\mathbf{y}^i = \mathbf{D}_m \mathbf{v}^i$, and $\mathbf{D}_m$ commutes with $\mathbf{M}_x$ for $m \in \mathbb{N}$ and $0 < x \le 1$. The latter is clear from the representations $\mathbf{D}_m = \mathbf{P}\mathbf{F}_m\mathbf{P}^{-1}$ and $\mathbf{M}_x = \mathbf{P}\mathbf{G}_x\mathbf{P}^{-1}$, with $\mathbf{P}_{N,k} = \binom{N}{k}$ and diagonal matrices $\mathbf{F}_m, \mathbf{G}_x$ defined by $(\mathbf{F}_m)_{N,k} = N^{\underline{m}}\delta_{N,k}$ and $(\mathbf{G}_x)_{N,k} = x^N\delta_{N,k}$. For the proof of the second assertion we note that in

$$-c\sum_{n\ge 0}\prod_{k=1}^{n}\mathbf{B}_k\mathbf{f}^n \preceq \mathbf{y}^0 = \sum_{n\ge 0}\prod_{k=1}^{n}\mathbf{B}_k\mathbf{x}^n \preceq c\sum_{n\ge 0}\prod_{k=1}^{n}\mathbf{B}_k\mathbf{f}^n$$

we can estimate the first $\bar{n} := \lfloor \frac{K}{K-u}\log_{1/p} N \rfloor$ terms of the right-hand side by

$$\prod_{k=1}^{n}\mathbf{B}_k\mathbf{f}^n = \mathbf{f} - \sum_{j=0}^{n-1}\prod_{k=1}^{j}\mathbf{B}_k\mathbf{e} \preceq \mathbf{f}.$$

For the remaining terms (for $n \ge \bar{n}$) we use $f_N^i = \mathcal{O}(N)$, which follows from (2), and property 2 of Lemma 1:

$$\sum_{n\ge\bar{n}}\prod_{k=1}^{n}\mathbf{B}_k\mathbf{f}^n \preceq c'\sum_{n\ge\bar{n}}\prod_{k=1}^{n}\mathbf{B}_k\mathbf{id} = c'\,\mathbf{id}\sum_{n\ge\bar{n}}\prod_{k=1}^{n}(a_{0,k}p + a_{1,k}q) \preceq \tfrac{c''}{N}\mathbf{id},$$

with $c', c'' > 0$ and $\mathbf{id}$ denoting the sequence $(N)_{N\ge 0}$. We conclude by obtaining $|y_N^0| \le c(\bar{n}f_N + c'') = \mathcal{O}(f_N \ln N)$. □

By Lemma 9, item 1 of Lemma 2, and (2) we have for $m \ge 0$

$$N^{\underline{m}}\Delta^m s_{N-m}^i = \mathcal{O}\left(\frac{(f_{N-m}^i)^2}{N}\right) = \mathcal{O}\left(\frac{(f_N^i)^2}{N}\right) = \mathcal{O}\left(f_N^i\right),$$

and hence Lemma 10 yields (36). The proof of the $\mathcal{O}$-part of Theorem 2 is thus complete.

In order to obtain the Ω result, we define the sequence $\underline{\mathbf{s}}$ via $\underline{s}_N := \inf_{i\ge 0} s_N^i$ and derive from (35)

$$\mathbf{v} \succeq (\mathbf{I} + \mathbf{B}_1 + \mathbf{B}_1\mathbf{B}_2 + \cdots)\underline{\mathbf{s}}.$$

Since $\underline{s}_N \ge 0$ for all $N \ge 0$ and $\underline{s}_2 > 0$, the latter because of $f_2^{i+1} \ge 1$ and

$$\begin{aligned} s_2^i &= \sum_{k=0}^{2}\binom{2}{k}p^k q^{2-k}\left(a_{0,i+1}(f_k^{i+1} - p^2 f_2^{i+1}) + a_{1,i+1}(f_{2-k}^{i+1} - q^2 f_2^{i+1})\right)^2 \\ &= (f_2^{i+1})^2 \times \begin{cases} p^2(1-p^2), & (a_{0,i+1}, a_{1,i+1}) = (1,0), \\ q^2(1-q^2), & (a_{0,i+1}, a_{1,i+1}) = (0,1), \\ (p^2+q^2)(1-p^2-q^2), & (a_{0,i+1}, a_{1,i+1}) = (1,1), \end{cases} \end{aligned}$$

we can prove $v_N = \Omega(f_N)$ by applying the following two lemmas.

LEMMA 11. *Let sequences* $\mathbf{p}$ *and* $\mathbf{b}$ *be given, with* $p_0 = b_0 = 1$ *and* $0 < p_i < 1$, $b_i > 0$ *for* $i \ge 1$, *and let* $\boldsymbol{\ell} := \sum_{i\ge 0} b_i \mathbf{M}_{p_i}\mathbf{e}$. *If there exists* $\varepsilon > 0$ *such that for all* $0 < x < 1$ *we have*

$$\sum_{i\ge 0} b_i \mathbb{1}_{]\varepsilon x, x]}(p_i) \ge \varepsilon \sum_{i\ge 0} b_i \mathbb{1}_{]x,\infty[}(p_i) \tag{41}$$

and

$$\sum_{i\ge 0} b_i p_i^2 \mathbb{1}_{]\varepsilon x, x]}(p_i) \ge \varepsilon \sum_{i\ge 0} b_i p_i^2 \mathbb{1}_{]0,\varepsilon x]}(p_i), \tag{42}$$

then for any sequence $\mathbf{x}$ *of nonnegative terms having at least one positive term and satisfying* $x_0 = x_1 = 0$, *the sequence* $\mathbf{y} := \sum_{i\geq 0} b_i \mathbf{M}_{p_i}\mathbf{x}$ *satisfies*

$$y_N = \Omega(\ell_N).$$

Proof. Let $k_0 \geq 2$ be the smallest k such that $x_k > 0$. With the help of (15) it is easily checked that

$$(\mathbf{M}_p \mathbf{e})_N \leq 1 \wedge N^2 p^2.$$

Thus for $N \geq k_0$ we have

$$\begin{aligned}
&\inf_{\varepsilon N^{-1} < p \leq N^{-1}} \binom{N}{k_0} p^{k_0}(1-p)^{N-k_0} \\
&\qquad = \binom{N}{k_0}\left[\left(\frac{\varepsilon}{N}\right)^{k_0}\left(1-\frac{\varepsilon}{N}\right)^{N-k_0} \wedge \left(\frac{1}{N}\right)^{k_0}\left(1-\frac{1}{N}\right)^{N-k_0}\right] \\
&\qquad \geq \binom{N}{k_0}\left[\left(\frac{\varepsilon}{N}\right)^{k_0} e^{-\varepsilon} \wedge \left(\frac{1}{N}\right)^{k_0} e^{-1}\right] \geq k_0^{-k_0}\left(\varepsilon^{k_0} e^{-\varepsilon} \wedge e^{-1}\right).
\end{aligned}$$

We define $C_{k_0,\varepsilon} := k_0^{k_0}\left(\varepsilon^{-k_0} e^{\varepsilon} \vee e\right)$. Using properties (41) and (42) with $x = \frac{1}{N}$, we obtain for $N \geq k_0$

$$\begin{aligned}
\ell_N &\leq \sum_{p_i \leq \frac{1}{N}} b_i (Np_i)^2 + \sum_{p_i > \frac{1}{N}} b_i \leq \left(\frac{1}{\varepsilon}+1\right) \sum_{\frac{\varepsilon}{N} < p_i \leq \frac{1}{N}} b_i (Np_i)^2 + \frac{1}{\varepsilon} \sum_{\frac{\varepsilon}{N} < p_i \leq \frac{1}{N}} b_i \\
&\leq \left(1+\frac{2}{\varepsilon}\right) \sum_{\frac{\varepsilon}{N} < p_i \leq \frac{1}{N}} b_i \leq \left(1+\frac{2}{\varepsilon}\right) C_{k_0,\varepsilon} \sum_{i\geq 0} \binom{N}{k_0} b_i p_i^{k_0} (1-p_i)^{N-k_0} \\
&\leq \left(1+\frac{2}{\varepsilon}\right) C_{k_0,\varepsilon} \frac{1}{x_{k_0}} \sum_{i\geq 0} b_i \mathbf{M}_{p_i} \mathbf{x}_N.
\end{aligned}$$

This completes the proof. □

Remark 10. For any $N \geq 2$ we have $\ell_N < \infty$ iff $\sum_{i\geq 0} b_i p_i^2 < \infty$, since $\frac{1}{4}(1 \wedge N^2 p^2) \leq (\mathbf{M}_p \mathbf{e})_N \leq 1 \wedge N^2 p^2$ for $N \geq 2$. Moreover, for $N \geq k_0$ we have $y_N < \infty$ iff $\ell_N < \infty$ by Lemma 11 and because of $y_N \leq \ell_N \max_{k\leq N} x_k$. If the sequence $\mathbf{p}$ has only one limit point, it follows from (41) that this limit point must be 0, and then from (42) that $\sum_{i\geq 0} b_i p_i^2 < \infty$ holds.

LEMMA 12. *Let* $\mathbf{p}$ *and* $\mathbf{b}$ *be as in Lemma* 11. *Then the conditions* (41) *and* (42) *of Lemma* 11 *are satisfied, and, moreover,* $\sum_{i\geq 0} b_i p_i^2 < \infty$ *holds, if the function* G, *defined by* $G(x) := -\sum_{i\geq 0} b_i \mathbb{1}_{]-\infty, p_i[}(x)$, *with values in* $[-\infty, 0]$, *has a representation*

$$G(x) = \sum_{n\geq 0} g_n(x),$$

with $g_0(x) = -\mathbb{1}_{]-\infty,1[}(x)$ *and* $g_n(x) = \sum_{i\geq 0} b_i^{(n)} g_{n-1}\left(\frac{x}{p_i^{(n)}}\right)$ *for* $n \geq 1$, *where the* $b_i^{(n)}$ *are nonnegative, the* $p_i^{(n)}$ *are positive and less than or equal to* 1, *and*

$$\inf_{n\geq 1} \sum_{i\geq 0} b_i^{(n)} > 1, \quad \sup_{n\geq 1} \sum_{i\geq 0} b_i^{(n)} \left(p_i^{(n)}\right)^2 < 1, \quad \text{and} \quad \inf_{n,i:\, b_i^{(n)} > 0} p_i^{(n)} > 0$$

hold.

Proof. We redefine, only for purposes of this proof, the symbol Δ: We let $\Delta f(x) = f(x) - f(x-)$ denote the jump of the right continuous function f at x. As we will shortly show,

$$\sum_{i\geq 0} b_i p_i^2 =: C < \infty \tag{43}$$

holds, so we derive for $x > 0$

$$G(x) = -\sum_{p_i > x} b_i \geq -\frac{1}{x^2} \sum_{p_i > x} b_i p_i^2 \geq -\frac{C}{x^2}.$$

Hence G is an increasing and right continuous step function on $]0,\infty[$, with values in $]-\infty, 0]$, and thus serves as a generalized distribution function (cf. [28, p. 158]) of a discrete measure μ on $]0,\infty[$ given by

$$\mu(]a,b]) := G(b) - G(a) = \sum_{a<t\leq b} \Delta G(t).$$

Defining another measure ν, absolutely continuous with respect to μ, by

$$\nu(]a,b]) := \sum_{a<t\leq b} t^2 \Delta G(t),$$

we have to prove the inequalities

$$\mu(]\varepsilon x, x]) \geq \varepsilon \mu(]x,\infty[), \qquad \nu(]\varepsilon x, x]) \geq \varepsilon \nu(]0, \varepsilon x]),$$

which are obtained by rephrasing (41) and (42) in terms of μ and ν. Moreover, (43) will be proved by showing $\nu(]0,\infty[) < \infty$. Therefore we denote

$$\alpha = \inf_{n\geq 1} \sum_{i\geq 0} b_i^{(n)}, \ \beta = \sup_{n\geq 1} \sum_{i\geq 0} b_i^{(n)} \left(p_i^{(n)}\right)^2, \ \text{ and } \ \eta = \inf_{n,i:\, b_i^{(n)}>0} p_i^{(n)} \tag{44}$$

and define measures μ_n, ν_n for $n \geq 0$ by

$$\mu_n(]a,b]) := g_n(b) - g_n(a), \quad \nu_n(]a,b]) := \sum_{a<t\leq b} t^2 \Delta g_n(t).$$

Now we observe that for $x > 0$

$$\mu_n(]x,\infty[) = -g_n(x) = -\sum_{i\geq 0} b_i^{(n)} g_{n-1}\left(\frac{x}{p_i^{(n)}}\right) \geq -\alpha g_{n-1}\left(\frac{x}{\eta}\right) = \alpha \mu_{n-1}(]x/\eta,\infty[),$$

which results in

$$\alpha\mu(]x,\infty[) = \alpha \sum_{n\geq 0} \mu_n(]x,\infty[) \leq \sum_{n\geq 1} \mu_n(]x\eta,\infty[) \leq \mu(]x\eta,\infty[)$$

and thus, since both sides are finite,

$$\mu(]x\eta, x]) \geq (\alpha - 1)\mu(]x,\infty[). \tag{45}$$

Similarly, we obtain

$$\begin{aligned}\nu_n(]0,x]) &= \sum_{0<t\le x} t^2\Delta g_n(t) = \sum_{i\ge0} b_i^{(n)} \sum_{0<t\le x} t^2\Delta g_{n-1}\left(\frac{t}{p_i^{(n)}}\right)\\ &= \sum_{i\ge0} b_i^{(n)} \left(p_i^{(n)}\right)^2 \sum_{0<t\le x/p_i^{(n)}} t^2\Delta g_{n-1}(t)\\ &\le \beta \sum_{0<t\le x/\eta} t^2\Delta g_{n-1}(t) = \beta\nu_{n-1}(]0,x/\eta]).\end{aligned}$$

Starting with $\nu_0(]0,\infty[) = 1$, induction yields $\nu_k(]0,\infty[) \le \beta^k$ and therefore

$$\sum_{i\ge0} b_i p_i^2 = \nu(]0,\infty[) = \sum_{n\ge0} \nu_n(]0,\infty[) \le \frac{1}{1-\beta} < \infty,$$

which proves (43), and, moreover,

$$\nu(]0,x\eta]) = \sum_{n\ge0} \nu_n(]0,x\eta]) \le \nu_0(]0,x\eta]) + \beta \sum_{n\ge0} \nu_n(]0,x]) = \nu_0(]0,x\eta]) + \beta\nu(]0,x]).$$

If $x < 1$, then also $x\eta < 1$ and thus $\nu_0(]0,x\eta]) = 0$, so we end up with

$$\nu(]x\eta,x]) \ge \frac{1-\beta}{\beta}\nu(]0,x\eta]). \tag{46}$$

The inequalities (45) and (46) remain true if we replace η, $\alpha-1$, and $\frac{1-\beta}{\beta}$ by $\varepsilon := \min\bigl(\eta, \alpha-1, \frac{1-\beta}{\beta}\bigr)$; thus the hypotheses (41) and (42) are valid. □

We cannot apply Lemma 12 directly since the sequences $\mathbf{b}^{(n)}, \mathbf{p}^{(n)}$ that would have to be chosen, namely, $b_i^{(n)} = a_{i,n}\mathbb{1}_{\{0,1\}}(i)$, $p_0^{(n)} = p$, $p_1^{(n)} = q$, do not satisfy the hypotheses. In the notation of (44) we have $\alpha = 1$, $\beta = p^2+q^2$, and $\gamma = q$, and thus the condition $\alpha > 1$ is violated. We thus recall (19) and apply Lemma 12 to the sequences $\mathbf{b}$ and $\mathbf{p}$ given by $\sum_{n\ge0}\prod_{k=1}^{Kn}\mathbf{B}_k = \sum_{i\ge0} b_i\mathbf{M}_{p_i}$. The pairs of sequences $\mathbf{b}^{(n)}, \mathbf{p}^{(n)}$ that we have to provide directly correspond to the products

$$\prod_{k=Kn-K+1}^{Kn} \mathbf{B}_k = (\mathbf{M}_p+\mathbf{M}_q)^u(\mathbf{M}_p)^{s_0}(\mathbf{M}_q)^{s_1},$$

where $0 \le s_0 \le K-u$ and $s_0+s_1 = K-u$, in which case $\mathbf{b}^{(n)}$ (resp., $\mathbf{p}^{(n)}$) is defined by $b_i^{(n)} = \binom{u}{i}$ (resp., $p_i^{(n)} = p^{i+s_0}q^{u-i+s_1}$) for $0 \le i \le u$, and these are easily seen to satisfy the conditions listed in Lemma 12. The estimate $v_N = \Omega(f_N)$ thus follows from

$$\sum_{n\ge0}\prod_{k=1}^{Kn} \mathbf{B}_k\underline{\mathbf{s}} \preceq \sum_{n\ge0}\prod_{k=1}^{Kn} \mathbf{B}_k\mathbf{s}^{Kn} \preceq \mathbf{v},$$

and the proof of Theorem 2 is complete. □

Proof of Corollary 2. It suffices to show that for each $\boldsymbol{\kappa} \in K_\omega$ we have P-almost surely $\frac{F_{\boldsymbol{\kappa},N}}{f_{\boldsymbol{\kappa},N}} \to 1$, as $N\to\infty$, since then also $P\times Q$-almost surely, as $N\to\infty$,

$$\ln C_N - \beta\ln N = \ln f_{\boldsymbol{\kappa},N} - \beta\ln N + o(1). \tag{47}$$

We will derive a stronger result: For fixed $\boldsymbol{\kappa} \in K_\omega$ and $\varepsilon > 0$ we have, P-almost surely, as $N \to \infty$,

$$\frac{F_{\boldsymbol{\kappa},N} - f_{\boldsymbol{\kappa},N}}{(f_{\boldsymbol{\kappa},N})^{2/3+\varepsilon}} \to 0. \tag{48}$$

We suppress $\boldsymbol{\kappa}$ and indeed obtain, by Theorem 2 and Chebyshev's inequality,

$$\mathbb{P}\left(|F_N - f_N| \geq (f_N)^{2/3+\varepsilon/2}\right) = \mathcal{O}\left((f_N)^{-1/3-\varepsilon} \ln N\right),$$

so that (48) is P-almost surely fulfilled for

$$N = N_k := \min(N \in \mathbb{N} : (f_N)^{1/3+\varepsilon} \geq k^{1+3\varepsilon})$$

and $k \to \infty$ by the Borel–Cantelli lemma. We readily derive

$$f_{N_{k+1}} - f_{N_k} \sim (k+1)^3 - k^3 \sim 3k^2 \leq 3(f_{N_{k+1}})^{2/3},$$

and $\sup_{k\geq 1} f_{N_{k+1}}/f_{N_k} < \infty$. Thus, if $|F_N - f_N| < (f_N)^{2/3+\varepsilon/2}$ holds for each $N \in \{N_k, N_{k+1}\}$, then for $N_k < n < N_{k+1}$ we obtain

$$|F_n - f_n| \leq |F_{N_k} - f_{N_{k+1}}| \vee |F_{N_{k+1}} - f_{N_k}| = \mathcal{O}\left((f_{N_{k+1}})^{\frac{2}{3}+\frac{\varepsilon}{2}}\right) = \mathcal{O}\left((f_n)^{\frac{2}{3}+\frac{\varepsilon}{2}}\right),$$

using the fact that the sequence $(f_k)_{k\geq 0}$ and the random sequence $(F_k)_{k\geq 0}$ are both increasing. This proves (48).

Next we show that there is no limiting distribution for C_N, no matter how we normalize it. We will rather consider $C_N N^{-\beta}$, which for $N \to \infty$-almost surely equals $\exp(\beta S_{\cdot,n} + \mathcal{O}(n^{0.4})) =: X_n$ with $n = \lfloor c_\beta \ln N \rfloor$; cf. (3) and (47). In order to obtain a nondegenerate limiting distribution for $\frac{X_n - b_n}{a_n}$, we should choose a_n at least of the order of the distance of the 0.25 and the 0.75 quantile of X_n, which is $e^{c\sqrt{n}+\mathcal{O}(n^{0.4})}$ with some $c > 0$. The median of X_n, which would be a good candidate for b_n, is of order $e^{\mathcal{O}(n^{0.4})} = o(a_n)$, therefore we can w.l.o.g. choose $b_n = 0$ for $n \geq 0$. Assume now that $Y_n := a_n^{-1} X_n \xrightarrow{\mathcal{D}} Y$ for some positive random variable Y which is not almost surely constant. By appropriate choice of $(a_n)_{n\geq 0}$ we can w.l.o.g. achieve that $\delta := \mathbb{P}(Y \in]0,1])$ satisfies $0 < \delta < 1$. Then

$$\mathbb{P}(Y_n \in]0,1]) = \mathbb{P}(\ln X_n \in]-\infty, \ln a_n]) =: \delta_n \to \delta,$$

but also

$$\begin{aligned}\mathbb{P}(Y_n \in]0,n]) &= \delta_n + \mathbb{P}(\ln X_n \in]\ln a_n, \ln a_n + \ln n]) \\ &= \delta_n + \mathbb{P}(\beta S_{\cdot,n} \in]\ln a_n - \mathcal{O}(n^{0.4}), \ln a_n + \mathcal{O}(n^{0.4})]) \\ &= \delta_n + \mathcal{O}(n^{-0.1}) \to \delta.\end{aligned}$$

Thus for any $x \geq 1$ we deduce $\mathbb{P}(Y \in]0,x]) = \delta$, which implies $\delta = 1$ and so contradicts our assumption on δ. □

Proof of Theorem 3. For the proof of the inequality $\alpha < \xi < \gamma_1$ we compare the corresponding defining equations, where we have to show

$$p^s > \left(p^{1+2s} + q^{1+2s}\right)^{\frac{1}{2}} > (p^{1+s} + q^{1+s}).$$

Now, the left inequality is a simple consequence of $q < p$, and the right inequality follows from the arithmetic-quadratic mean inequality.

For the result concerning $\operatorname{Var} C_N$ we first observe

$$\begin{aligned}\operatorname{Var} C_N &= \operatorname{Var} f_{\boldsymbol{\kappa},N} + \mathbb{E}\, v_{\boldsymbol{\kappa},N}\\ &= \mathbb{E}\,(f_{\boldsymbol{\kappa},N})^2 - (\mathbb{E}\, f_{\boldsymbol{\kappa},N})^2 + \mathcal{O}(N^{\alpha}\ln N)\\ &= \mathbb{E}\,(f_{\boldsymbol{\kappa},N})^2 + \mathcal{O}(N^{2\alpha}),\end{aligned}$$

where all the expectations and variances on the right-hand side are with respect to Q. From Lemma 5 we know that

$$f_{\boldsymbol{\kappa},N} \asymp \sum_{n\ge 0}\sum_{k=0}^{un}\binom{un}{k}\left(1\wedge\left(N\rho_{_{Kn}}p^{un-k}q^{k}\right)^2\right) = e^{\mathcal{O}(\ln\ln N)}\max_{n_0\le n\le n_2}\binom{un}{\lfloor k_n\rfloor},$$

from which we deduce

$$(f_{\boldsymbol{\kappa},N})^2 = e^{\mathcal{O}(\ln\ln N)}\sum_{n\ge 0}\sum_{k=0}^{un}\binom{un}{k}^2\left(1\wedge\left(N\rho_{_{Kn}}p^{un-k}q^{k}\right)^4\right),$$

and furthermore $\mathbb{E}\,(f_{\boldsymbol{\kappa},N})^2 = e^{\mathcal{O}(\ln\ln N)}\Psi(N)$, where

$$\Psi(N) = \sum_{n\ge 0}\sum_{j=0}^{(K-u)n}\binom{(K-u)n}{j}p^{(K-u)n-j}q^{j}\sum_{k=0}^{un}\binom{un}{k}^2\left(1\wedge\left(Np^{Kn-k-j}q^{k+j}\right)^4\right).$$

The asymptotics of $\Psi(N)$ will now be determined by invoking its Mellin transform $\Psi^*(s)$, which will be seen to exist for $-4 < \Re s < -2\xi$.

$$\Psi^*(s) := \int_0^\infty N^{s-1}\Psi(N)dN = \int_0^\infty N^{s-1}(1\wedge N^4)dN\cdot\Lambda(s) = -\frac{4}{s(s+4)}\Lambda(s),$$

where, with the abbreviation $a_s = (p^{1-s}+q^{1-s})^{K/u-1}$,

$$\begin{aligned}\Lambda(s) &= \sum_{n\ge 0}\sum_{j=0}^{(K-u)n}\binom{(K-u)n}{j}(p^{(K-u)n-j}q^{j})^{1-s}\sum_{k=0}^{un}\binom{un}{k}^2(p^{un-k}q^{k})^{-s}\\ &= \sum_{n\ge 0}a_s^{un}[z^{un}](1+p^{-s}z)^{un}(1+q^{-s}z)^{un}\\ &= \frac{1}{2\pi i}\oint_{|z|=(pq)^{\Re s/2}}\frac{dz/z}{1-a_s^u(1+p^{-s}z)^u(1+q^{-s}z)^u z^{-u}}\\ &= \frac{1}{u}\sum_{k=1}^{u}\frac{1}{2\pi i}\oint_{|z|=(pq)^{\Re s/2}}\frac{dz}{z-\varepsilon_k a_s(1+p^{-s}z)(1+q^{-s}z)}\\ &= \frac{1}{u}\sum_{k=1}^{u}\frac{1}{\sqrt{\left(1-\varepsilon_k a_s(p^{-s}+q^{-s})\right)^2-4\varepsilon_k^2a_s^2(pq)^{-s}}}\\ &= \frac{1}{u}\sum_{k=1}^{u}\frac{1}{\sqrt{(1-\varepsilon_k a_s(p^{-s/2}+q^{-s/2})^2)(1-\varepsilon_k a_s(p^{-s/2}-q^{-s/2})^2)}}.\end{aligned}$$

Here $(\varepsilon_k)_{k=1}^u$ denotes the set of the uth roots of unity. Next, for $\Re s < -2\xi$ we have $|\varepsilon_k a_s(p^{-s/2}\pm q^{-s/2})^2| < a_{-2\xi}(p^{\xi}+q^{\xi})^2 = 1$ by the definition of ξ. Thus $\Psi^*(s)$ is

analytic for $\Re s < -2\xi$, with the exception of a simple pole at $s = -4$. Moreover it can be shown that for some $\delta > 0$ all the zeros of the functions $1-\varepsilon_k a_s(p^{-s/2}\pm q^{-s/2})^2$ belonging to the strip $-2\xi \le \Re s \le -2\xi + \delta$ are simple and uniformly discrete in the sense that distances between distinct zeros are lower bounded by a positive constant; cf. [26, Lemma 4]. The function $\Lambda(s)$ is analytic in a domain which is cut along rays that start at the singularities and have the direction of the positive real axis. We can therefore recover the asymptotics of $\Psi(N)$ by applying the Cauchy integral theorem to the inverse Mellin integral

$$\Psi(N) = \frac{1}{2\pi i}\int_{-2-i\infty}^{-2+i\infty} \Psi^*(s) N^{-s} ds,$$

moving the line of integration to the right, thereby obtaining a curve which is composed of vertical pieces and Hankel-type pieces encircling the singularities of $\Lambda(s)$ in the strip $-2\xi \le \Re s \le -2\xi + \delta$. One of the main terms (and in the case $\frac{\ln p}{\ln q} \notin \mathbb{Q}$ the only main term; cf. [26, Lemma 4]) of the asymptotics of $\Psi(N)$ is contributed by the singularity at $s = -2\xi$. There we have the expansion

$$\Psi^*(s) = \frac{d_{-1}}{\sqrt{-s-2\xi}} + d_0 + \mathcal{O}\left(\sqrt{-s-2\xi}\right),$$

with positive d_{-1}. Now, for $r > 0$ and $r + \delta' \le \delta$, let the clockwise-oriented curve $\mathcal{C}$ consist of the piece $\{te^{2\pi i} : r \le t + 2\xi \le r + \delta'\}$ "below" the cut, the circle $\{-2\xi + re^{i\phi} : 0 \le \phi \le 2\pi\}$ and the piece $\{t : r \le t + 2\xi \le r + \delta'\}$ "above" the cut. Making use of Hankel's integral representation of $1/\Gamma(s)$, we obtain

$$\frac{1}{2\pi i}\int_{\mathcal{C}} \Psi^*(s) N^{-s} ds = \frac{N^{2\xi}}{\sqrt{\pi \ln N}}\left(d_{-1} + \mathcal{O}\left(\frac{1}{\ln N}\right)\right).$$

In the case $\frac{\ln p}{\ln q} \in \mathbb{Q}$ there will be infinitely many singularities with real part -2ξ and imaginary parts that are integer multiples of some positive constant, whose total contribution is $\frac{N^{2\xi}}{\sqrt{\pi \ln N}}\tau(\ln N) + \mathcal{O}(N^{2\xi}\ln^{-3/2} N)$, where τ is a continuous periodic function of mean d_{-1}. The other singularities in the strip $-2\xi \le \Re s \le -2\xi + \delta$ and the vertical pieces of the contour contribute $o(N^{2\xi}\ln^{-1/2} N)$. Since $\Psi(N)$ is increasing, the periodic function τ has to be strictly positive, so we indeed obtain $\Psi(N) = N^{2\xi}e^{\mathcal{O}(\ln\ln N)}$, which completes the proof. □

5. The limiting distribution for fixed queries.

Proof of Theorem 4. Throughout this proof, dependencies on $\boldsymbol{\kappa}$, which is fixed, will be suppressed. A central limit theorem for martingale difference arrays will be employed. To show that the two hypotheses of that theorem are satisfied, we will make heavy use of Lemmas 9 and 10 from the preceding section. Some notation of the first paragraph of section 3 will be used, and in particular we note that

$$\{\nu(m) : m \ge 2\} = \{0.\nu(m) : m \ge 1\} \cup \{1.\nu(m) : m \ge 1\},$$

with the dot denoting concatenation. For instance, $\nu(10) = 010 = 0.\nu(6)$ implies $\mathsf{t}^{10} = \mathsf{t}_\ell^6$.

Our first goal is to represent each $(F_N^i - f_N^i)/\sqrt{v_N^i}$ as the terminal value of some martingale. We observe that $X_{N,m} := |\mathsf{t}^{2m}|$ is a random variable on T_N for $m \ge 1$. Moreover we let $X_{N,0} \equiv N$ and define for $m \ge 0$ sigma algebras

$$\mathcal{F}_{N,m} = \sigma(X_{N,j},\, 0 \le j \le m),$$

in particular $\mathcal{F}_{N,0} = \{\emptyset, \mathsf{T}_N\}$. This gives rise to a filtration $\mathbb{F}_N := (\mathcal{F}_{N,m})_{m\geq 0}$. For $m \geq 1$ a wordy description of $\mathcal{F}_{N,m}$ would be that it accumulates knowledge of the sizes of the $2m$ subtrees of a trie t built from N keys that are rooted in the two sons of any of the nodes v_j for $1 \leq j \leq m$.

For $i \geq 0$ and $m \geq 1$ we define functions on $\bigcup_{N\geq 0} \mathsf{T}_N$ by

$$\lambda_m^i(\mathsf{t}) := \varepsilon_m^i \left(1 + a_{0,i+1+\ell(m)} f_{|\mathsf{t}^{2m}|}^{i+1+\ell(m)} + a_{1,i+1+\ell(m)} f_{|\mathsf{t}^{2m+1}|}^{i+1+\ell(m)} - f_{|\mathsf{t}^m|}^{i+\ell(m)}\right) \mathbb{1}_{\{|\mathsf{t}^m|\geq 2\}}$$

and immediately note that

$$\lambda_m^i(\mathsf{t}) = \varepsilon_m^i \lambda_1^{i+\ell(m)}(\mathsf{t}^m),$$

where $\ell(m) := \lfloor \log_2 m \rfloor$ denotes the level of node v_m, and ε_m^i is defined in (8). We let $\lambda_{N,m}^i$ be the restriction of λ_m^i to T_N and claim

(i) $f_N^i + \sum_{m\geq 1} \lambda_{N,m}^i = F_N^i$,
(ii) $\lambda_{N,m}^i$ is $\mathcal{F}_{N,m}$-measurable,
(iii) $\mathbb{E}\left[\lambda_{N,m}^i \middle| \mathcal{F}_{N,m-1}\right] = 0$.

For the proof of (i) we observe, using $\varepsilon_{2m}^i = \varepsilon_m^i a_{0,i+1+\ell(m)}$ and $\varepsilon_{2m+1}^i = \varepsilon_m^i a_{1,i+1+\ell(m)}$, that the terms involving f form a telescoping series, and what remains is exactly the series given in (9). Since $|\mathsf{t}^m|$, $|\mathsf{t}^{2m}|$, and $|\mathsf{t}^{2m+1}| = |\mathsf{t}^m| - |\mathsf{t}^{2m}|$ are all $\mathcal{F}_{N,m}$-measurable, so is $\lambda_{N,m}^i$, which proves (ii). Finally we turn to (iii), which is true, since for any fixed $n \geq 2$ and for $k \sim B(n,p)$ we have

$$\begin{aligned}&\mathbb{E}\left[\lambda_{N,m}^i \middle| |\mathsf{t}^m| = n\right] \\ &\quad = \varepsilon_m^i \mathbb{E}\left[1 + a_{0,i+1+\ell(m)} f_k^{i+1+\ell(m)} + a_{1,i+1+\ell(m)} f_{n-k}^{i+1+\ell(m)} - f_n^{i+\ell(m)}\right] = 0\end{aligned}$$

by (10). We continue defining

$$\xi_{N,m}^i := \frac{\lambda_{N,m}^i}{\sqrt{v_N^i}}$$

and observe that the sequence $\left(\sum_{m=1}^M \xi_{N,m}^i\right)_{M\geq 1}$ is a martingale with respect to the filtration $\mathbb{F}_N$, which converges P_N-almost surely and in $L^2(\mathsf{T}_N, \mathcal{T}_N, P_N)$ to the terminal value

$$\frac{F_N^i - f_N^i}{\sqrt{v_N^i}} = \sum_{m=1}^{\infty} \xi_{N,m}^i.$$

By a well-known central limit theorem for martingale difference arrays [14, 28] we have, as $N \to \infty$,

$$\sum_{m=1}^{\infty} \xi_{N,m}^i \xrightarrow{\mathcal{D}} \mathcal{N}(0,1)$$

if both the "conditional normalizing condition"

$$\text{(No)} \qquad \sum_{m=1}^{\infty} \mathbb{E}\left[(\xi_{N,m}^i)^2 | \mathcal{F}_{N,m-1}\right] \xrightarrow{P} 1 \text{ as } N \to \infty$$

and the "conditional Lindeberg condition"

$$\text{(Li)} \qquad \sum_{m=1}^{\infty} \mathbb{E}\left[(\xi^i_{N,m})^2 \mathbb{1}_{\{|\xi^i_{N,m}|>\varepsilon\}} \middle| \mathcal{F}_{N,m-1}\right] \xrightarrow{P} 0 \text{ as } N \to \infty \quad \text{ for all } \varepsilon > 0$$

are satisfied.

It is easy to derive sufficient conditions for (No) and (Li): Let

$$V^i_N := \sum_{m=1}^{\infty} \mathbb{E}\left[(\lambda^i_{N,m})^2 | \mathcal{F}_{N,m-1}\right]$$

be the terminal value of the predictable quadratic variation process of the martingale $\left(\sum_{m=1}^{M} \lambda^i_{N,m}\right)_{M\geq 0}$. As we will shortly show, V^i_N satisfies the following equation, which is reminiscent of (6):

$$V^i_N \overset{\mathcal{D}}{=} \mathbb{1}_{\{N\geq 2\}} \left(s^i_N + a_{0,i+1} V^{i+1}_k + a_{1,i+1} \bar{V}^{i+1}_{N-k}\right),$$

where $k \sim B(N,p)$, $V^i_k \overset{\mathcal{D}}{=} \bar{V}^i_k$, and $V^i_k, \bar{V}^i_{N-k}$ are independent, conditional on k, and where

$$s^i_N = \mathbb{E}\left[(\lambda^i_{N,1})^2 | \mathcal{F}_{N,0}\right] = \operatorname{Var}\left(a_{0,i+1} f^{i+1}_k + a_{1,i+1} f^{i+1}_{N-k}\right)$$

was introduced in (34). We first derive

$$\mathbb{E}\left[(\lambda^i_{N,m})^2 | \mathcal{F}_{N,m-1}\right] = \mathbb{E}\left[(\varepsilon^i_m \lambda^{i+\ell(m)}_{|\mathsf{t}^m|,1})^2 | \mathcal{F}_{|\mathsf{t}^m|,0}\right] = \varepsilon^i_m \mathbb{E}\left(\lambda^{i+\ell(m)}_{|\mathsf{t}^m|,1}\right)^2 = \varepsilon^i_m s^{i+\ell(m)}_{|\mathsf{t}^m|}.$$

Now, for $m \geq 2$, let the numbers $\alpha \in \{0,1\}$ and $\mu \in \mathbb{N}$ be uniquely defined by $\nu(m) = \alpha.\nu(\mu)$. Then $\varepsilon^i_m s^{i+\ell(m)}_{|\mathsf{t}^m|} = a_{\alpha,i+1} \varepsilon^{i+1}_\mu s^{i+1+\ell(\mu)}_{|(\mathsf{t}^{2+\alpha})^\mu|}$ holds, which implies

$$\begin{aligned} V^i(\mathsf{t}) :&= \sum_{m\geq 1} \varepsilon^i_m s^{i+\ell(m)}_{|\mathsf{t}^m|} = s^i_{|\mathsf{t}|} + a_{0,i+1} \sum_{m\geq 1} \varepsilon^{i+1}_m s^{i+1+\ell(m)}_{|\mathsf{t}_\ell{}^m|} + a_{1,i+1} \sum_{m\geq 1} \varepsilon^{i+1}_m s^{i+1+\ell(m)}_{|\mathsf{t}_r{}^m|} \\ &= s^i_{|\mathsf{t}|} + a_{0,i+1} V^{i+1}(\mathsf{t}_\ell) + a_{1,i+1} V^{i+1}(\mathsf{t}_r); \end{aligned} \tag{49}$$

hence, with V^i_N of course the restriction of V^i to T_N, the above distributional equation for V^i_N readily follows.

Condition (No) demands $\frac{V^i_N}{v^i_N} \xrightarrow{P} 1$, which is implied by

$$\text{(No}_2\text{)} \qquad \operatorname{Var} V^i_N = o\left((v^i_N)^2\right).$$

Note that by (49) we have

$$\operatorname{Var} F^i_N = v^i_N = \mathbb{E}\, V^i_N = \mathbb{E} \sum_{m\geq 1} \varepsilon^i_m s^{\ell(m)+i}_{|\mathsf{t}^m|},$$

and thus similarly (i.e., by constructing a martingale with respect to $\mathbb{F}_N$ converging to V^i_N, and considering its predictable quadratic variation process) we obtain

$$\operatorname{Var} V^i_N = \mathbb{E} \sum_{m\geq 1} \varepsilon^i_m \sigma^{\ell(m)+i}_{|\mathsf{t}^m|},$$

where

$$\sigma_N^i = \operatorname{Var}\left(a_{0,i+1}v_k^{i+1} + a_{1,i+1}v_{N-k}^{i+1}\right).$$

In (Li) we fix $a > 1$ and use $x^2 \mathbb{1}_{\{|x|>\varepsilon\}} \le \frac{x^{2a}}{\varepsilon^{2(a-1)}}$ to construct another sequence of random variables

$$\Lambda_N^i \overset{\mathcal{D}}{=} \mathbb{1}_{\{N\ge 2\}}\left(\tilde{s}_N^i + a_{0,i+1}\Lambda_k^{i+1} + a_{1,i+1}\bar{\Lambda}_{N-k}^{i+1}\right),$$

where $k \sim B(N,p)$, $\Lambda_k^i \overset{\mathcal{D}}{=} \bar{\Lambda}_k^i$, and Λ_k^i, $\bar{\Lambda}_{N-k}^i$ are independent, conditional on k, and where

$$\tilde{s}_N^i = \mathbb{E}\left[(\lambda_{N,1}^i)^{2a}|\mathcal{F}_{N,0}\right].$$

Condition (Li) is thus implied by

$$\mathbb{E}\,\Lambda_N^i = o\left((v_N^i)^a\right). \tag{Li$_a$}$$

We obtain equations of the same type as (11) and (35) for $w_N := \operatorname{Var} V_N^0$ and $y_N := \mathbb{E}\,\Lambda_N^0$, namely,

$$\mathbf{w} = \sum_{n\ge 0}\prod_{i=1}^n \mathbf{B}_i\boldsymbol{\sigma}^n, \qquad \mathbf{y} = \sum_{n\ge 0}\prod_{i=1}^n \mathbf{B}_i\tilde{\mathbf{s}}^n,$$

and will show $w_N = o((v_N)^2)$ and $y_N = o((v_N)^a)$. Using the expansion (38) of f_k^i around $k = pN$ (we can choose $M = 0$), we obtain $\tilde{s}_N^i = \mathcal{O}\big(\frac{(f_N^i)^{2a}}{N^a}\big)$, which will be $\mathcal{O}(f_N^i)$ if $a > 1$ is only small enough. With the help of Lemma 9 we obtain $\Delta^m s_N^i = \mathcal{O}\big(\frac{f_N^i}{N^m}\big)$, and thus upon application of Lemma 10 also $\Delta^m v_N^i = \mathcal{O}\big(\frac{v_N^i}{N^m}\ln N\big)$. Now an expansion of v_k^i around $k = pN$ yields $\sigma_N^i = \mathcal{O}\big(\frac{(v_N^i)^2}{N}\big) = \mathcal{O}\big(\frac{(f_N^i)^2\ln^2 N}{N}\big) = \mathcal{O}\big(f_N^i\big)$. We have thus proven $\sigma_N^i = \mathcal{O}(f_N^i)$ and $\tilde{s}_N^i = \mathcal{O}(f_N^i)$, and it follows from Lemma 10 that both w_N and y_N are of order $\mathcal{O}(f_N \ln N)$. On the other hand, $(v_N)^a = \Omega((f_N)^a)$ for $a \ge 0$. Since $\ln N = o\left((f_N)^{a-1}\right)$ for any $a > 1$ by (2), we see that (No$_2$) and (Li$_a$) indeed hold. □

6. Conclusion. While costs C_N of partial match retrievals in K-d tries constructed from N records have been investigated in depth under the symmetric Bernoulli model (sBm), no results going beyond asymptotics of expected costs seem to be available under the asymmetric Bernoulli model (aBm). It has been the aim of the present paper to fill this gap. For aBm a distinction between fixed and random queries has to be made which is not necessary for sBm. In the random query model it turns out that for aBm we have $(\mathbb{E}\,C_N)^2 = o(\mathbb{E}\,C_N^2)$, in contrast to sBm, where $(\mathbb{E}\,C_N)^2 \sim \mathbb{E}\,C_N^2$, and due to cancellations, $\operatorname{Var} C_N = \Theta(\mathbb{E}\,C_N)$ emerges. Furthermore for sBm the normalized cost $\frac{C_N - \mathbb{E}\,C_N}{\sqrt{\operatorname{Var} C_N}}$ converges in distribution to a standard normal random variable. This does not hold for aBm, where there is no nondegenerate limiting distribution at all. However, in this case $\ln C_N$, properly normalized, follows a central limit law. Differences between sBm and aBm are not so drastic for the fixed query model. Here $\operatorname{Var} C_N = \Theta(\mathbb{E}\,C_N)$ and $\frac{C_N - \mathbb{E}\,C_N}{\sqrt{\operatorname{Var} C_N}} \overset{\mathcal{D}}{\to} \mathcal{N}(0,1)$ are true also for aBm, but, depending on the query, $\mathbb{E}\,C_N$ may now enjoy a growth between $\Omega(N^{\gamma_2})$ and $\mathcal{O}(N^{\gamma_1})$ and will almost surely be "close" to N^β, for certain constants

$\gamma_2 < \beta < \gamma_1$, where closeness is expressed in terms of a law of the iterated logarithm. In particular, expected growth and almost sure growth are not the same for aBm.

The paper considers only alphabets of size 2, but we are quite sure that our results can without much pain be generalized to analogous statements holding for larger alphabets.

Acknowledgment. I wish to thank the referees for their detailed reading and constructive comments on the first version of this paper.

REFERENCES

[1] J. L. Bentley, *Multidimensional binary search trees used for associative retrieval*, Comm. ACM, 18 (1975), pp. 509–517.
[2] W. A. Burkhard, *Hashing and trie algorithms for partial match retrieval*, ACM Trans. Database Systems, 1 (1976), pp. 175–187.
[3] P. Chanzy, L. Devroye, and C. Zamora-Cura, *Analysis of range search for random k-d trees*, Acta Inform., 37 (2001), pp. 355–383.
[4] H.-H. Chern and H.-K. Hwang, *Partial match queries in random quadtrees*, SIAM J. Comput., 32 (2003), pp. 904–915.
[5] H. Chernoff, *Estimation of the mode*, Ann. Inst. Statist. Math., 16 (1964), pp. 31–41.
[6] M. Csörgö and L. Horváth, *Weighted Approximations in Probability and Statistics*, Wiley Ser. Probab. Math. Statist., Wiley-Interscience, Chichester, UK, 1993.
[7] L. Devroye, J. Jabbour, and C. Zamora-Cura, *Squarish k-d trees*, SIAM J. Comput., 30 (2000), pp. 1678–1700.
[8] A. Duch, V. Estivill-Castro, and C. Martínez, *Randomized K-dimensional binary search trees*, in Algorithms and Computation (Taejon, 1998), Lecture Notes in Comput. Sci. 1533, Springer, Berlin, 1998, pp. 199–208.
[9] P. Flajolet, X. Gourdon, and P. Dumas, *Mellin transforms and asymptotics: Harmonic sums*, Theoret. Comput. Sci., 144 (1995), pp. 3–58.
[10] P. Flajolet and C. Puech, *Partial match retrieval of multidimensional data*, J. ACM, 33 (1986), pp. 371–407.
[11] E. Fredkin, *Trie memory*, Comm. ACM, 3 (1960), pp. 490–500.
[12] P. Groeneboom, *Brownian motion with a parabolic drift and Airy functions*, Probab. Theory Related Fields, 81 (1989), pp. 79–109.
[13] W. Hoeffding, *On the distribution of the number of successes in independent trials*, Ann. Math. Statist., 27 (1956), pp. 713–721.
[14] P. Hall and C. C. Heyde, *Martingale Limit Theory and Its Application*, Academic Press, New York, 1980.
[15] P. Jacquet and M. Régnier, *Normal limiting distribution of the size of tries*, in Proceedings of Performance 87, North–Holland, Amsterdam, 1988, pp. 209–223.
[16] P. Kirschenhofer and H. Prodinger, *Multidimensional digital searching—alternative data structures*, Random Struct. Algorithms, 5 (1994), pp. 123–134.
[17] P. Kirschenhofer, H. Prodinger, and W. Szpankowski, *Multidimensional digital searching and some new parameters in tries*, Internat. J. Found. Comput. Sci., 4 (1993), pp. 69–84.
[18] D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, MA, 1998.
[19] H. M. Mahmoud, *Evolution of Random Search Trees*, Wiley, New York, 1992.
[20] C. Martinez, A. Panholzer, and H. Prodinger, *Partial match queries in relaxed multidimensional search trees*, Algorithmica, 29 (2001), pp. 181–204.
[21] C. McDiarmid, *Concentration*, in Probabilistic Methods for Algorithmic Discrete Mathematics, M. Habib et al., eds., Algorithms Comb. 16, Springer, Berlin, 1998, pp. 195–248.
[22] R. Neininger, *Asymptotic distributions for partial match queries in K-d trees*, Random Struct. Algorithms, 17 (2000), pp. 403–427.
[23] R. Neininger and L. Rüschendorf, *Limit laws for partial match queries in quadtrees*, Ann. Appl. Probab., 11 (2001), pp. 452–469.
[24] R. L. Rivest, *Partial-match retrieval algorithms*, SIAM J. Comput., 5 (1976), pp. 19–50.
[25] W. Schachinger, *The variance of a partial match retrieval in a multidimensional symmetric trie*, Random Struct. Algorithms, 7 (1995), pp. 81–95.
[26] W. Schachinger, *Limiting distributions for the costs of partial match retrievals in multidimensional tries*, Random Struct. Algorithms, 17 (2000), pp. 428–459.

[27] W. Schachinger, *Concentration of size and path length of tries*, Combin. Probab. Comput., to appear.
[28] A. N. Shiryaev, *Probability*, 2nd ed., Springer, New York, 1996.
[29] D. Steinsaltz, *Socks & Boxes: Variations on Daniel Bernoulli's Marriage Problem*, Ph.D. thesis, Harvard University, Cambridge, MA, 1996.
[30] W. Szpankowski, *Average Case Analysis of Algorithms on Sequences*, Wiley, New York, 2001.

SIAM J. COMPUT.
Vol. 33, No. 4, pp. 984–1009

ON PROVING CIRCUIT LOWER BOUNDS AGAINST THE POLYNOMIAL-TIME HIERARCHY*

JIN-YI CAI† AND OSAMU WATANABE‡

Abstract. We consider the problem of proving circuit lower bounds against the polynomial-time hierarchy. We give both positive and negative results. For the positive side, for any fixed integer $k > 0$, we give an explicit Σ_2^{p} language, acceptable by a Σ_2^{p} machine with running time $O(n^{k^2+k})$, that requires circuit size $> n^k$. This provides a constructive version of an existence theorem of R. Kannan [*Inform. and Control*, 55 (1982), pp. 40–56]. Our main theorem is on the negative side. We give evidence that it is infeasible to give relativizable proofs that any single language in the polynomial-time hierarchy requires superpolynomial circuit size. Our proof techniques are based on the decision tree version of the Switching Lemma for constant depth circuits and the Nisan–Wigderson pseudorandom generator. We also take this opportunity to publish some previously unpublished older results of the first author on constant depth circuits, both straight lower bounds and inapproximability results based on decision tree–type Switching Lemmas.

Key words. stringent relativization, circuit complexity, computational complexity, Switching Lemma, Nisan–Wigderson generator

AMS subject classifications. 68Q15, 68Q17

DOI. 10.1137/S0097539703422716

1. Introduction.

It is a most basic open problem in theoretical computer science to give circuit lower bounds for various complexity classes. The class P has polynomial-size circuits. It is also widely believed that NP does not share this property, i.e., that some specific set such as SAT in NP requires superpolynomial circuit size. While this remains the most concrete approach to the NP vs. P problem, we can't even prove, for any fixed $k > 1$, that any set $L \in \mathrm{NP}$ requires circuit size $> n^k$.

If we relax the restriction from NP to the second level of the Polynomial-time Hierarchy[1] Σ_2^{p}, R. Kannan [Kan82] did prove that for any fixed polynomial n^k, there is some set L in Σ_2^{p} which requires circuit size $> n^k$. Kannan in fact proved the existence theorem for some set in $\Sigma_2^{\mathrm{p}} \cap \Pi_2^{\mathrm{p}}$. This result has been improved by Köbler and Watanabe [KW98], who showed, based on the technique developed in [BCGKT], that such a set exists in $\mathrm{ZPP}^{\mathrm{NP}}$. More recently, the work in [Cai01] implies that a yet lower class $\mathrm{S}_2^{\mathrm{p}}$ contains such a set. (See [BFT98, MVW99] for related topics.)

However, Kannan's proof for Σ_2^{p}, and all the subsequent improvements mentioned above, are not "constructive" in the sense that it does not identify a single Σ_2^{p} machine whose language requires circuit size $> n^k$. In this paper we first remark on this point and give some constructive proofs for Σ_2^{p}.

*Received by the editors February 13, 2003; accepted for publication (in revised form) February 10, 2004; published electronically May 25, 2004.
http://www.siam.org/journals/sicomp/33-4/42271.html

†Computer Science Department, University of Wisconsin, Madison, WI 53706 (jyc@cs.wisc.edu). The work of this author was supported in part by NSF grants CCR-0208013 and CCR-0196197 and by U.S.-Japan Collaborative Research NSF SBE-INT 9726724.

‡Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Tokyo 152-8552, Japan (watanabe@is.titech.ac.jp). The work of this author was supported in part by JSPS/NSF Collaborative Research 1999 and by the Ministry for Education, Grant-in-Aid for Scientific Research (C), 2001.

[1]We will use standard notions and notation of complexity theory; see textbooks, e.g., [DK00], for definitions.

At the top level, all these proofs for the above-mentioned results are of the same type. Let us review Kannan's proof for Σ_2^{p}: *Either* SAT does not have n^k size circuits, and then we are done, *or* SAT has n^k size circuits, and then we can define some other set, which by the existence of the hypothetical circuit for SAT can be shown in Σ_2^{p}, and it requires circuit size $> n^k$. Thus, in each case, we have some set L_0 in Σ_2^{p} that has no n^k size circuits; but this does not give any single Σ_2^{p} machine whose language requires circuit size $> n^k$. Constructively, Kannan gave a set in $\Sigma_4^{\mathrm{p}} \cap \Pi_4^{\mathrm{p}}$. In [MVW99] a set in Δ_3^{p} was constructively given. We improve this to Σ_2^{p}.

THEOREM 1. *For any integer $k > 0$, we can construct a Σ_2^{p} machine with $O(n^{k^2} \log^{k+1} n)$ running time that accepts a set with no n^k size circuits.*

Notice that Σ_2^{p} has complete languages. Thus, by using any standard complete language C for Σ_2^{p}, it is easy to obtain a result like the above. We can argue as follows: Estimate the time complexity of a reduction from L_0 to C, which is possible even from the above "nonconstructive" proof of the existence of L_0. Then define a padded version C' of C so that L_0 is reducible to C in linear time. This C' is a language that requires circuit size $> n^k$; clearly, we can give explicitly a Σ_2^{p} machine recognizing C' and its time bound. Our contribution here is to show a way to construct such a machine directly, which we hope to be of any help when discussing a similar constructive proof that is open for the stronger statements, i.e., the existence of a set with n^k circuit size lower bound in $\Sigma_2^{\mathrm{p}} \cap \Pi_2^{\mathrm{p}}$ (resp., $\mathrm{ZPP}^{\mathrm{NP}}$ and $\mathrm{S}_2^{\mathrm{p}}$).

Our main result in this paper deals with the difficulty in proving superpolynomial circuit size lower bound for any set in the Polynomial-time Hierarchy, PH. While it is possible to prove, for any fixed $k > 0$, that Σ_2^{p} has a set with circuit size $> n^k$, the real challenge is to prove a superpolynomial circuit size lower bound for a single language. Not only have we not been able to do this for any set in NP, but also no superpolynomial lower bound is known for any set in PH. In this paper we prove that it is infeasible to give relativizable superpolynomial lower bound for any set in the Polynomial-time Hierarchy.

For our relativized argument, we propose a new computation model that gives us more "stringent" relativized results. Relativization results can be generally classified as either separation or collapsing/containment results. The implication of a relativized separation result is that the corresponding collapse is difficult to prove. Similarly a relativized collapsing result implies that the corresponding separation is difficult to prove. Notice that it is still possible (and, in fact, such examples have been shown) to have a separation or collapsing result against the corresponding relativized result; relativized results just suggest some "difficulty" and not "impossibility." Also we should note that the degree of "difficulty" may depend on a relativization type. Here we deal with relativized collapsing results, and we introduce a new relativization notion—stringent relativization—for demonstrating the difficulty of proving circuit lower bounds for PH.

By surveying existing relativized collapsing results, we came to realize that an asymmetry is often present. In almost all of these relativized collapsing results the proof is achieved by allowing stronger access to oracles by the simulating computation than the simulated computation. For example, in the usual proof of $\mathrm{P}^A = \mathrm{NP}^A$ or $\mathrm{P}^A = \mathrm{PSPACE}^A$, we encode QBF in the oracle. In terms of the simulation by the $\mathrm{P}^{\mathrm{QBF}}$ machine $\mathcal{M}$ simulating an $\mathrm{NP}^{\mathrm{QBF}}$ or $\mathrm{PSPACE}^{\mathrm{QBF}}$ computation $\mathcal{M}'$ on an input x, $\mathcal{M}$ will access an oracle location polynomially longer than the corresponding access that $\mathcal{M}'$ makes. That is, P^A machines are given more powerful oracle access. One can argue that this asymmetry is within a polynomial factor, but it nonetheless denies

access to certain segments of the oracle to the simulated machine while affording such access to the simulating machine. In our present study of specific polynomial bounds such as n^k of either circuit size or running time, this arbitrary polynomial stretch in oracle access is not acceptable.

We study in this paper some specific circuit size lower bound with respect to some specific, say n^k, time bound. Then, as the above observation suggests, we must adopt the following more "stringent" oracle computation model. In this more "stringent" oracle access model, we require that, for any input, the simulating machine or circuit does not access the oracle of length longer than the simulated machine or circuit can access on this input. We consider circuits consisting of standard AND, OR, and NOT gates and oracle query gates. An oracle query gate takes m input bits $z = b_1 b_2 \dots b_m$ and has output $\chi_{[z \in X]}$, i.e., it outputs 1 or 0 depending on whether $z \in X$ or otherwise. Now the proviso of "stringent access to an oracle" is stated as follows: To show that, at length n, a circuit C_n recognizes the language of machine $\mathcal{M}$ with running time n^k, we allow the circuit access only to those strings of length $\leq n^k$. For any machine $\mathcal{M}$, we say that a family of circuits $\{C_n\}_{n \geq 0}$ *simulates* $\mathcal{M}^X$ under a *stringent access* to the oracle X if at every length n, C_n recognizes $L(\mathcal{M}^X)^{=n}$ by stringent access to the oracle X.

Though we defined our stringency notion as above for the comparison between machines and circuits with polynomial-time and polynomial-size resource bounds, we believe that the notion of "stringent oracle access" is meaningful in a more general setting. (For more general situations, it might be better to consider a more robust notion of "stringent relativization." We leave this issue and more general investigations of "stringent relativization" for our future work; see, e.g., [CW04].)

From a more general perspective, the main utility of relativization is to show the inadequacy of certain proof techniques. Certainly the more "stringent" a requirement we place on the type of relativization, the stronger the result will be, and perhaps it says more about the infeasibility of certain techniques. Imagine there are three possible claims of proving a certain lower bound, such as Σ_2^{p} requires superpolynomial circuit size:

1. A proof totally specific to a specific set in Σ_2^{p} that uses specific properties of the circuit combinatorics.
2. A proof for a general Σ_2^{p} machine that uses properties of the circuit, but which can be carried through if the machine and circuit were allowed to access any but the same segment of any oracle, i.e., they could ask queries within the same length bound. (In our case, the length bound is defined as the time bound of the simulated Σ_2^{p} machine.)
3. A proof more general, for a general Σ_2^{p} machine and circuit, and which can go through even if we allowed the machine and circuit to access different segments of the oracle.

Any relativization to the contrary says nothing about the first possibility. A relativization to the contrary with stringent access model rules out possibilities 2 and 3. If we did not have the "stringent access requirement," then we can only rule out 3, but not 2.

In this paper we focus on specific time/size bounds. In the stringent oracle access model, we prove that for any alternating oracle TM $\mathcal{M}$ with running time $O(n^k)$, there is an oracle X and a polynomial-size circuit family accepting it. Therefore we rule out possibilities 2 and 3 above.

THEOREM 2 (main theorem). *For any integer $d > 0$ and any real $k > 1$, let $\mathcal{M}$*

be an oracle Σ_d^{p} machine with running time $O(n^k)$. Then we have an oracle X and a family of Boolean circuits $\{C_n\}_{n\geq 0}$ that recognizes $L(\mathcal{M}^X)$ under a stringent access *to the oracle X. For all sufficiently large n, the size of C_n is bounded by n^{cdk} for some universal constant $c > 0$.*

From this, we can conclude that, relative to the oracle X given above, every set in PH has some polynomial-size circuits, i.e., $\mathrm{PH}^X \subseteq \mathrm{P}^X/\mathrm{poly}$. Recall that Heller [He84] showed an oracle Y such that $\mathrm{EXP}^Y \subseteq \mathrm{P}^Y/\mathrm{poly}$, which immediately implies that $\mathrm{PH}^Y \subseteq \mathrm{P}^Y/\mathrm{poly}$. But this oracle Y is not used in a stringent way; that is, a circuit simulating a given Σ_d^{p} machine $\mathcal{M}$ on inputs of length n makes queries to Y that are longer than the time bound of $\mathcal{M}$ on length n inputs. (Notice that our stringency condition is for polynomial-time bounds. It would be possible to extend it to exponential-time bounds, and we may claim that the oracle Y is used in a stringent way in the relation $\mathrm{EXP}^Y \subseteq \mathrm{P}^Y/\mathrm{poly}$, because any polynomial bound for circuit size is less than exponential-time bounds of simulated EXP machines. In this sense, our stringency notion is not robust, and we need a more robust notion for general investigations; see, e.g., [CW04].)

Our proof technique for the main theorem is based on the decision tree version of the Switching Lemma for constant depth circuits and the Nisan–Wigderson pseudorandom generator.

As these results crucially depend on lower bounds for constant depth circuits, we take this opportunity to publish some unpublished older results of the first author on constant depth circuits, as it would fit the theme. These include both straight lower bounds and inapproximability results based on decision tree–type Switching Lemmas. We give some better constants in the exponents than previously published lower bounds.

2. Proof of Theorem 1. R. Kannan [Kan82] proved that for any fixed polynomial n^k, there is some set L in $\Sigma_2^{\mathrm{p}} \cap \Pi_2^{\mathrm{p}}$ with circuit size $> n^k$. However, in terms of explicit construction, he only gave a set in $\Sigma_4^{\mathrm{p}} \cap \Pi_4^{\mathrm{p}}$. An improvement to Δ_3^{p} was stated in [MVW99].

In this section we give a constructive proof of Kannan's theorem for Σ_2^{p}.

For any $n \geq 0$, a binary sequence χ of length $\ell \leq 2^n$ is called a *partial characteristic sequence*, which will specify lexicographically the membership of the first ℓ strings of $\{0,1\}^n$. We denote this subset of $\{0,1\}^n$ by $L(\chi)$. We say that χ is consistent with a circuit C with n input gates iff $\forall i$, $1 \leq i \leq \ell$, $C(x_i)$ outputs the ith bit of χ, where x_i is the ith string of $\{0,1\}^n$.

We can encode every circuit C of size $\leq s$ as a string u of length $\mathrm{len}(s)$, where $\mathrm{len}(s)$ is defined as $\mathrm{len}(s) = c_{\mathrm{circ}} \lfloor s \log s \rfloor$ with some constant c_{circ}. We may consider every u with $|u| = \mathrm{len}(s)$ encodes some circuit of size $\leq s$; if a string u is not a proper code or the encoded circuit has size $> s$, we assume that this u encodes the constant 0 circuit. The following lemma is immediate by counting.

LEMMA 3. *For any $s > 1$, there exists a partial characteristic sequence of length $\ell = \mathrm{len}(s) + 1$ that is not consistent with any circuit of size $\leq s$.*

Our goal is to define a set L that has no n^k size circuit but that is recognized by some explicitly defined Σ_2^{p} machine. Our construction follows essentially the same outline as the one given in [MVW99], which in turn uses ideas given in Kannan's original proof. The further improvement is mainly an even more efficient use of alternation.

For a given n, let $\ell = \mathrm{len}(n^k) + 1$. We try to construct a partial characteristic sequence χ_{non} of length ℓ that is consistent with *no* circuit of size $\leq n^k$. We will

introduce an auxiliary set PreCIRC that is in NP. With this PreCIRC, some Σ_2^{p} machine can *uniquely* determine the desired characteristic sequence χ_{non} (on its accepting path). We would like to define our set L (partially) consistent with this sequence χ_{non}. But Σ_2^{p} computation using some auxiliary NP set cannot be implemented, in general, by any Σ_2^{p} machine. Suppose here that PreCIRC has n^k size circuits; then some Σ_2^{p} machine can guess such circuits, verify them, and use them for computing χ_{non} and recognizing strings according to χ_{non}. What if there are no such circuits for PreCIRC? We will define L so that one part of L is consistent with PreCIRC (while the other part is consistent with χ_{non} if PreCIRC is computable by some n^k size circuits). If PreCIRC has no n^k size circuit, then the part of L that is consistent with PreCIRC can guarantee the desired hardness of L.

Now we describe our construction in detail. We fix any sufficiently large n and let $\ell = \mathrm{len}(n^k) + 1$. By "$v \succ u$" we mean that u is a prefix of v. To compute the "hard" characteristic sequence χ_{non}, we want to determine, for a given pair of a partial characteristic sequence χ and a string u, whether u can be extended to some description v of a circuit that is consistent with χ. The set PreCIRC is defined for this task. More precisely, for any $n > 0$, and for any strings χ of length ℓ and u of length $\leq \mathrm{len}(n^k)$, we define PreCIRC as follows.

$$1^n 0 \chi u 0 1^{\mathrm{len}(n^k)-|u|} \in \mathrm{PreCIRC}$$
$$\Leftrightarrow\ (\exists v \succ u)\ [\ |v| = \mathrm{len}(n^k), \text{ and the circuit encoded by } v \text{ is consistent with } \chi\ ].$$

Strings of any other form are not contained in PreCIRC. For simplifying our notation, we will simply write (χ, u) for $1^n 0 \chi u 0 1^{\mathrm{len}(n^k)-|u|}$. Since n determines ℓ, and the length of χ is ℓ, χ and u are uniquely determined from $0^n 1 \chi u 1 0^{\mathrm{len}(n^k)-|u|}$. The length of (χ, u) is $\widetilde{n} = n + 2\ell + 1$. Note that $\widetilde{n}$ is $O(n^k \log n)$.

We now define our machine $\mathcal{M}$. Informally we want $\mathcal{M}$ to accept an input x iff either $x \in 1\{0,1\}^{n-1}$ and $x \in \mathrm{PreCIRC}$, or $x \in 0\{0,1\}^{n-1}$ and $x \in L$, where $L^{=n}$ is a set with no n^k size circuits, for all sufficiently large n, *if* $\mathrm{PreCIRC}^{=n}$ has n^k size circuits for all sufficiently large n. Specifically, $\mathcal{M}$ is designed so that $L^{=n}$ would be $L(\chi_{\mathrm{non}})$, where χ_{non} is lexicographically the first χ of length ℓ with no n^k size circuit, *provided* $\mathrm{PreCIRC}^{=\widetilde{n}}$ has an $\widetilde{n}^k$ size circuit for length $\widetilde{n}$. Note that $L(\chi_{\mathrm{non}}) \subseteq 0\{0,1\}^{n-1}$ since $|\chi_{\mathrm{non}}| = \mathrm{len}(n^k) + 1 < 2^{n-1}$.

More formally, for any given input x of length n, if x starts with 1, then $\mathcal{M}$ accepts it iff $x \in \mathrm{PreCIRC}$. Suppose otherwise; that is, x starts with 0. Then first $\mathcal{M}$ existentially guesses a partial characteristic sequence χ_{non} of length ℓ *and* a circuit C_{pre} of size $\widetilde{n}^k$, more precisely, a string v_{pre} of length $\mathrm{len}(\widetilde{n}^k)$ encoding a circuit for $\mathrm{PreCIRC}^{=\widetilde{n}}$ of size $\leq \widetilde{n}^k$. (Below we use C_{pre} to denote the circuit that is encoded by the guessed v_{pre}.) After that, $\mathcal{M}$ enters the universal stage, where it checks the following items.

(1) C_{pre} makes no mistake whenever it says "yes": $\forall \chi$, $|\chi| = \ell$, and $\forall u$, $|u| \leq \mathrm{len}(n^k)$, verify that C_{pre} is "locally consistent" on (χ, u) if $C_{\mathrm{pre}}(\chi, u) = 1$; that is, check as follows:

$$C_{\mathrm{pre}}(\chi, u) = 1 \ \&\ |u| = \mathrm{len}(n^k) \implies \text{the circuit that } u \text{ encodes is consistent with } \chi,$$
$$C_{\mathrm{pre}}(\chi, u) = 1 \ \&\ |u| < \mathrm{len}(n^k) \implies \text{either } C_{\mathrm{pre}}(\chi, u0) = 1 \text{ or } C_{\mathrm{pre}}(\chi, u1) = 1.$$

(2) C_{pre} says "yes" for all positive instances: $\forall u$, $|u| = \mathrm{len}(n^k)$, compute the χ_u of length ℓ defined by (the circuit encoded by) u, and verify that $C_{\mathrm{pre}}(\chi_u, u') = 1$ for every prefix u' of u.

(3) The guessed χ_{non} is lexicographically the first string of length ℓ such that no circuit of size s is consistent with it, *according to* C_{pre}: Check $C_{\text{pre}}(\chi_{\text{non}}, \epsilon) = 0$, and $\forall \chi$ such that $|\chi| = \ell$ and χ is lexicographically smaller than χ_{non}, check that $C_{\text{pre}}(\chi, \epsilon) = 1$ holds. (Here ϵ denotes the empty string.)

Finally on each universal branch, if $\mathcal{M}$ passes the particular test of this branch, then $\mathcal{M}$ accepts the input $x \in 0\{0,1\}^{n-1}$ iff χ_{non} has bit 1 for the string x.

For all (χ, u), such that $|\chi| = \ell$ and $|u| \leq \text{len}(n^k)$, if C_{pre} passes item (1), then

$$C_{\text{pre}}(\chi, u) = 1 \implies (\chi, u) \in \text{PreCIRC},$$

and if C_{pre} passes item (2), then

$$(\chi, u) \in \text{PreCIRC} \implies C_{\text{pre}}(\chi, u) = 1.$$

Of course there is no guarantee that there exists a circuit C_{pre} (more precisely, v_{pre}) that will pass the tests in items (1) and (2). But if there is such a C_{pre}, then some existential path leads to such a C_{pre} together with the right χ_{non}. This χ_{non} is the lexicographically first string of length ℓ such that no circuit of size n^k is consistent with it, which exists by Lemma 3. In particular it is independent of the particular C_{pre} guessed. Hence, if there is a circuit for $\text{PreCIRC}^{=\widetilde{n}}$ of size $\widetilde{n}^k$, then $\mathcal{M}$ accepts $x \in 0\{0,1\}^{n-1}$ iff x is in $L(\chi_{\text{non}})$, where χ_{non} is the unique lexicographically first string of length ℓ with no consistent circuit of size $\leq n^k$.

On the other hand, if no circuit of size $\widetilde{n}^k$ can accept $\text{PreCIRC}^{=\widetilde{n}}$ correctly, then no circuit passes the tests in items (1) and (2), and hence $\mathcal{M}$ simply rejects all $x \in 0\{0,1\}^{n-1}$. But since $\text{PreCIRC}^{=\widetilde{n}}$ has no $\widetilde{n}^k$ size circuit, the hardness is guaranteed by the PreCIRC part of $L(\mathcal{M})$. More formally, if $\text{PreCIRC}^{=n}$ has no circuit of size n^k *infinitely often*, then we are done. Otherwise, for all sufficiently large n, and hence for all sufficiently large $\widetilde{n}$, a circuit C_{pre} exists for $\text{PreCIRC}^{=\widetilde{n}}$ of size $\widetilde{n}^k$; then the part $L(\mathcal{M}) \cap 0\{0,1\}^{n-1}$ is defined so that no n^k size circuit can accept it correctly, and hence again we are done. Therefore, we can conclude that $L(\mathcal{M})$ has no n^k size circuit (which by definition means that for infinitely many n this is so). This Σ_2^{p} language proves Theorem 1. It can be easily checked that the machine $\mathcal{M}$ runs in $O(n^{k^2} \log^{k+1} n)$ steps.

3. Proof of Theorem 2. We first give an outline of the proof.

3.1. Proof outline. Consider any Σ_d^{p} polynomial-time bounded oracle alternating Turing machine $\mathcal{M}$, with time bound n^k. We want to design an oracle X so that some family of small size circuits can simulate $\mathcal{M}^X$ with stringent oracle access. More specifically, for any fixed sufficiently large n, we want a circuit $C_{\mathcal{M}}$ that simulates $\mathcal{M}^X$ on inputs of length n, where, since $\mathcal{M}$ can query strings only of length at most n^k, we require that $C_{\mathcal{M}}$ can also only ask queries of length at most n^k.

It is well known from [FSS81] that a Σ_d^{p} machine $\mathcal{M}$ bounded in time n^k with oracle X, when given an input x of length n, gives rise to a bounded depth Boolean circuit C_x of the following type: The inputs are Boolean variables, and their negations, representing membership of a string $z \in \{0,1\}^{\leq n^k}$ in the oracle X. The Boolean circuit C_x starts with an OR gate at the top and alternates with AND's and OR's with depth $d+1$, where the bottom level gates have bounded fan-in at most n^k, and all other AND and OR gates are unbounded fan-in, except by the overall circuit size, which is bounded by $n^k 2^{n^k}$. Without loss of generality we may assume the Boolean circuit

is tree-like, except for the input level, where each Boolean variable corresponding to $\chi_{[z \in X]}$ is represented by a pair of complemented variables, which we will denote by z and $\overline{z}$.

Our first idea is to use random restrictions to "kill" the circuit. By this we mean that for any circuit C over Boolean variables $x_1, \ldots, x_n$, a *random restriction* ρ (for some specified parameter p) is a random function that assigns each x_i either 0, 1, or $*$, with probability $\Pr[\rho(x_i) = *] = p$ and $\Pr[\rho(x_i) = 0] = \Pr[\rho(x_i) = 1] = (1-p)/2$ for each i independently. Assigning $*$ means to leave it as a variable. Let $C\mid_\rho$ denote a circuit obtained by this random restriction. It is known that after a random restriction ρ (for a suitably chosen parameter p), the circuit $C\mid_\rho$ is sufficiently weakened so as to have either small min-terms or small max-terms. Results of this type are generally known as Switching Lemmas, and the strongest form known is due to Håstad [Hås86a]. (See also [Ajt83, FSS81, Yao85, Cai86, Hås86b].) However it turns out that we need a different form, namely, a decision tree–type Switching Lemma [Cai86]. We want to assign a suitably chosen random restriction ρ, after which the circuit admits a small depth decision tree. We in fact will have to consider an aggregate of 2^n such Boolean circuits C_x simultaneously, one each for an input x of size n. We want to assign ρ, after which all these circuits have small depth decision trees. We then will proceed to set those variables to ensure that all these circuits are "killed"; i.e., they all have a definite value now, either 0 or 1. We need to assign those variables consistently over all 2^n small depth decision trees. For decision trees, it is easy to achieve this by always setting "the next variable" asked by the decision tree to 0, say; it is not clear how to maintain this consistency in terms of min-terms and max-terms. If each decision tree has depth bounded by t, then we will have assigned at most $2^n t$ many variables corresponding to those strings of length n^k where ρ initially assigned an $*$ (i.e., they are left unassigned by ρ). We will argue that there are still plenty of unassigned variables left, where we may try to encode the now-determined computational values of these 2^n circuits. We will argue that t is sufficiently small, and yet with high probability all 2^n circuits admit decision trees of depth at most t.

The problem with this idea is that after we have coded the values of all the 2^n circuits in X, there does not seem to be any easy way to recover this information. Since X had already been "ravaged" by the random restriction ρ, it is not clear how to distinguish those "code bits" from those "random bits." Further complicating the matter are those bits assigned during the decision tree settlement. All of this must be sorted out, supposedly, by a polynomial-size oracle circuit which is to accept $L(\mathcal{M}^X)^{=n}$. Note that, after a random restriction ρ, it is probabilistically almost impossible to have an easily identifiable segment of the set X all assigned $*$ by ρ (e.g., all strings in $\{0,1\}^{=n^k}$ with a certain leading bit pattern), not to mention the subsequent all 0 assignment to fix the decision trees. On the other hand, we have 2^n computations to code. It is infeasible for the final polynomial-size oracle circuit to "remember" more than a polynomial number of bits as the address of the coding region. So it appears that we must have an easily identifiable region to code, identified with at most a polynomial number of bits for its address, and, to accommodate 2^n computations, this region must be large.

To overcome this difficulty, our idea is to use not true random restrictions, but rather pseudorandom restrictions via the Nisan–Wigderson generator [Nis91a, Nis91b, NW88]. Nisan and Wigderson designed a pseudorandom generator (which we will call an *NW generator*) provably indistinguishable from true random bits by polynomial-size constant depth circuits. While our circuits are not of polynomial size, this can

be scaled up easily. Our idea is then to use the output of some NW generator to perform the "random" restriction, and to argue that all 2^n circuits are "killed" with high probability, just as before with true random restrictions. The basic argument is that no constant depth circuits of an appropriate size can tell the difference under either a true random assignment or a pseudorandom assignment coming from the NW generator. However, for our purpose in this paper, we wish to say that a certain behavior of these 2^n constant depth circuits—namely, they are likely to possess small depth decision trees after a "random" restriction with 0's, 1's, and $*$'s—is preserved when "pseudorandom restrictions" are substituted for "random restrictions." It is vitally important that whatever property we wish to claim was maintained by the substitution of random bits by pseudorandom bits, the property must be expressible as a constant depth circuit with an appropriate size upper bound. It is not clear whether the property of "having a small depth decision tree" can be expressed in this way.

We overcome this difficulty by using a weaker property which is a consequence of "having a small depth decision tree," which nonetheless is sufficient for our purpose. Namely, we take directly the property that, after a restriction with 0's, 1's, and $*$'s, every one of the 2^n circuits can be determined by assigning additionally 0's to a small number of variables, which had been assigned $*$'s. This property is expressible in a constant depth way. Then we will mimic the probability distribution of the 0's, 1's, and $*$'s under the random restrictions by uniform random bits 0's and 1's, so that we can come up with a constant depth circuit D with the following property: It takes only Boolean inputs Ω of 0's and 1's, and D evaluates to 1 iff when a restriction ρ_Ω with 0's, 1's, and $*$'s defined by Ω is applied to all 2^n circuits C_x, every C_x can be set to either 0 or 1 after a small number of additional variables are set to 0. We will design D in such a way that under a uniform bit sequence Ω, D will almost certainly evaluate to 1.

In fact we need more than that. We also need to have the property that a certain segment of the oracle is untouched by the additional setting of 0's in all 2^n decision tree settlements. We will argue by the pigeonhole principle that our bounds guarantee a suitable region unspoiled by all these decision tree settlement variables. It is not reasonable to expect that any such region is entirely assigned with $*$'s, but at least there should be many $*$'s.

Assume now we have designed such a D satisfying all these requirements. For this D we apply the NW generator, substituting pseudorandom bits for true random bits Ω given to D as inputs. We conclude that D still evaluates to 1 with high probability. In particular, there must be some setting of the source bits ω for the generator, such that D is evaluated to 1. This implies that we can assign the oracle set X first according to the pseudorandom restriction described by the pseudorandom bits, then according to the 2^n small depth decision trees, which are guaranteed by the evaluation of D, and set these additional variables all to 0. This settles all the decision trees, and thus the values of all 2^n circuits C_x are determined. Furthermore, there is a significant segment T_{y_0} of X free from any variables used in any decision tree settlement, where we will code these 2^n results of C_x.

Even though this segment T_{y_0} is free from any variables used in any decision tree settlement, in order to code the computation results of C_x, there must be plenty of $*$'s left, and they must be recoverable by polynomial-size circuits. We will show that with high probability over a uniformly chosen random seed ω, the pseudorandom restriction defined by ω will leave plenty of $*$'s in each segment such as T_{y_0}. We then in fact choose a sequence of bits ω that satisfies both the requirement $D = 1$ and this additional requirement.

Finally, we will show that with a suitable choice of parameters in the *combinatorial design* used in the NW generator, we will be able to recover in polynomial time the location where we assigned $*$'s in X, in particular from within the coding segment of X given the address of this segment. Now our polynomial-size circuit $C_{\mathcal{M}}$ is designed as follows: It remembers (is hardwired with) y_0, i.e., the address of T_{y_0}, and remembers the seed ω for the NW generator, which is of polynomial length. Then on any input x, it performs the polynomial-time computation over a finite field to extract the coded result of C_x from the appropriate location in X.

3.2. Proof detail. Now we specify the parameters and state our proof precisely.

Fix any Σ_d^{p} polynomial-time bounded oracle alternating Turing machine $\mathcal{M}$, with time bound n^k. For notational convenience we will assume $k > 2$ and $d \geq 7$. We assume that n is sufficiently large. On input of length n, $\mathcal{M}$ can query strings only of length at most n^k. We will use m to denote n^k and M to denote 2^m throughout this proof.

Assume that membership in the oracle set has already been decided for all strings of length less than m. Our task is to fix the membership for "$z \in X$?" of length exactly m in X, so that for each input x of length n, membership "$x \in L(\mathcal{M}^X)$?" can be decided by a polynomial-size circuit $C_{\mathcal{M}}$ with oracle gates that can access $X^{=m}$. Since $X^{<m}$ has already been fixed, membership "$x \in L(\mathcal{M}^X)$?" is determined by the set $X^{=m}$. Here we specifically require that the circuit $C_{\mathcal{M}}$ can access only those strings that can be possibly accessed by the simulated machine $\mathcal{M}$ on input of length n.

There are 2^n inputs x of length n, and each computation of $\mathcal{M}$ on x gives rise to a depth $d+1$ Boolean circuit C_x with bottom fan-in at most m. The inputs to each circuit C_x are the $2M$ literals z and $\overline{z}$, where $z \in \{0,1\}^m$ corresponds to the truth value of $\chi_{[z \in X]}$. (To simplify our notation, we will denote by z both a string in $\{0,1\}^m$ as well as the Boolean variable corresponding to $\chi_{[z \in X]}$.) As stated earlier, we assume that each circuit C_x is a tree, starting with an OR gate at the top, and alternating with AND's and OR's until inputs z's and $\overline{z}$'s, where these inputs are duplicated to keep the tree structure. That is, each circuit C_x is a depth $d+1$ tree with size at most mM and bottom fan-in at most m.

A Switching Lemma shows that such a constant depth circuit is sufficiently weakened, after a suitably chosen random restriction ρ, so as to have either small min-terms or small max-terms. The strongest form known is due to Håstad [Hås86a]. For our purpose in this paper, however, we will require something more.

The decision tree complexity of a Boolean function f, denoted by $\mathrm{DC}(f)$, is the smallest depth of a Boolean decision tree computing the function. It can be shown easily that if $\mathrm{DC}(f) \leq t$, then f can be expressed both as an AND of OR's as well as an OR of AND's, with bottom fan-in at most t. Moreover, clearly, there is a subset of no more than t variables, such that if one assigns all of them to 0, the function f will be determined. This is an important advantage as we will have to assign many nondisjoint subsets of variables for multiple Boolean functions, and all these assignments need to be consistent.

A decision tree version of the Switching Lemma was first proved in [Cai86], where a different terminology, i.e., Master-Player Game and t-monochromaticity, was introduced. Adapting Håstad's proof to the decision tree model, one can prove the following lemma. In section 4, we will discuss these lemmas more thoroughly. See Remark 4 at the end of the paper for more background discussions.

LEMMA 4. *For any depth $d+1$ Boolean circuit C on M inputs $z_1, z_2, \ldots, z_M$, of*

size at most s and bottom fan-in at most t, we have

$$\Pr_\rho[\mathrm{DC}(C\mid_\rho) \geq t] \;\leq\; \frac{s}{2^t},$$

where the random restriction ρ is defined for $p = \frac{1}{(10t)^d}$.

To reduce all circuits C_x, $x \in \{0,1\}^n$, to small depth decision trees, we apply a random restriction with $p = 1/(20m)^d$ to these circuits. Then by the union bound we have the following.

Claim 1.

$$\Pr_\rho\left[\bigvee_{x\in\{0,1\}^n} [\mathrm{DC}(C_x\mid_\rho) \geq 2m]\right] \;\leq\; 2^n \cdot \frac{mM}{2^{2m}} \;=\; \frac{m}{2^{m-n}}.$$

That is, with probability close to 1, a random restriction reduces *every* circuit C_x to a decision tree of depth $< 2m$.

Below we will carry out a sequence of transformations on the circuits C_x, $x \in \{0,1\}^n$, with the ultimate goal of constructing the circuit D, which, in some sense, is a test for the success of a "random restriction."

Step 1 (C^1_x). C^1_x takes $2M$ Boolean inputs (a_z, b_z) for $z \in \{0,1\}^m$. The pair (a_z, b_z) will represent the status of the Boolean variable z to C_x as follows: $a_z = 1$ iff the value of z is set (to either 0 or 1, i.e., not set to $*$), and $a_z = 0$ otherwise. If $a_z = 1$, then the 0-1 value of z is represented by b_z. If the pair (a_z, b_z) represents the value of z, then the pair $(a_z, \overline{b_z})$ represents that of $\overline{z_i}$. Clearly, if z is set 0 (resp., 1), then $\overline{z}$ must be set 1 (resp., 0).

C^1_x is constructed from C_x as follows. Each gate g in C_x will be represented by a pair of gates (g_s, g_v). $g_s = 1$ iff g is set to either 0 or 1, i.e., it is determined; $g_s = 0$ otherwise. If $g_s = 1$, then $g = g_v$. Thus, $(g_s, g_v) = (0,0)$ or $(0,1)$ represent the situation where g has not been determined, and $(g_s, g_v) = (1,0)$ or $(1,1)$, respectively, represent the case where g is set to 0 or 1, respectively.

Suppose g is an OR gate, $g = \bigvee_{i=1}^s g^{(i)}$, where $g^{(i)}$ is an input literal or an internal gate. Suppose $g^{(i)}$ is represented by the pair $(g^{(i)}_s, g^{(i)}_v)$. This representation is already defined inductively. Then we let

$$g_s \;=\; \bigvee_{i=1}^s \left((g^{(i)}_s \wedge g^{(i)}_v)\right) \vee \left(\bigwedge_{i=1}^s (g^{(i)}_s \wedge \overline{g^{(i)}_v})\right).$$

That is, g is set iff either some g_i is set to 1 or else all g_i are set to 0. Note that the formula given for g_s is a depth 2 circuit of size $O(s)$. Also let

$$g_v \;=\; \bigvee_{i=1}^s g^{(i)}_v.$$

Note that g_v is a "valid" value for g only when $g_s = 1$. Also g_v is depth 1 and has size s.

The case $g = \bigwedge_{i=1}^s g^{(i)}$ is dual. In this case, g is set iff either some g_i is set to 0 or else all g_i are set to 1. Thus

$$g_s \;=\; \bigvee_{i=1}^s \left((g^{(i)}_s \wedge \overline{g^{(i)}_v})\right) \vee \left(\bigwedge_{i=1}^s (g^{(i)}_s \wedge g^{(i)}_v)\right) \quad \text{and} \quad g_v \;=\; \bigwedge_{i=1}^s g^{(i)}_v.$$

Again they are depth 2, size $O(s)$, and depth 1, size s, respectively.

In order to maintain alternating forms of AND's and OR's in the circuit C_x^1, with all negations pushed to the input level, we can represent each gate g by both g and its negated value $\overline{g}$. This can introduce at most a factor of 2 in the size. (In fact we will define only three gates g_s, g_v, and $\overline{g_v}$ in our construction; we do not need $(\overline{g})_s$. But we will omit the detailed analysis of constant factors.) C_x^1 has two output gates g_s and g_v for the output gate g of C_x. It follows that

$$\text{size}(C_x^1) = O(\text{size}(C_x)) \text{ and } \text{depth}(C_x^1) = 2\ \text{depth}(C_x).$$

We can take the constant in $O(\text{size}(C_x))$ to be 10, say.

Step 2 (C_x^2). Let $p = \frac{1}{(2m)^d}$. Let $L = \lceil \log_2 \frac{1}{p} \rceil \approx dk \log_2(20n)$. C_x^2 takes Boolean inputs $(a_{z,1}, \ldots, a_{z,L}, b_z)$ for $z \in \{0,1\}^m$. The circuit C_x^2 is identical to C_x^1, except instead of taking inputs a_z, it has $a_z = \bigvee_{j=1}^{L} a_{z,j}$.

Note that a random restriction ρ with parameter $\Pr[\rho(z) = *] = 1/2^L$ on C_x is simulated by uniformly and independently assigning all the bits $(a_{z,1}, \ldots, a_{z,L}, b_z)$ to 0 or 1, in C_x^2, for $z \in \{0,1\}^m$. The behavior of C_x is represented in C_x^2 exactly. Here we have

$$\text{size}(C_x^2) = \text{size}(C_x^1) + O(ML) \text{ and } \text{depth}(C_x^2) = \text{depth}(C_x^1) + 1.$$

Note also that $2^{-L} \leq p$. The same upper bound in Lemma 4 and Claim 1 still applies when ρ has parameter 2^{-L}.

Step 3 (C_x^3). In C_x^3 we will check for the existence of a subset $S \subset [M]$ of cardinality $|S| = 2m$ such that, first, they are assigned $*$ by the ρ, and, second, if we further set them all to 0, it would determine the circuit C_x. We know from Claim 1 that this is almost certainly true for our random restriction.

Thus, we let

$$C_x^3 \ = \ \bigvee_S \left[\ \bigwedge_{z \in S} \overline{a_z} \wedge [(C_x^2)_s]_S \ \right],$$

where $\bigvee_S$ ranges over all subsets $S \subset [M]$ of cardinality $|S| = 2m$, and $(C_x^2)_s$ is the "set bit output" for C_x^2, and $[(C_x^2)_s]_S$ is obtained from $(C_x^2)_s$ by setting all $b_z = 0$ for $z \in S$. Recall that $a_z = \bigvee_{j=1}^{L} a_{z,j}$. Then we have

$$\text{size}(C_x^3) \leq \binom{M}{2m} (\text{size}(C_x^2) + O(m)) \text{ and } \text{depth}(C_x^3) = \text{depth}(C_x^2) + 2.$$

Step 4 (D). Finally, define D by

$$D \ = \ \bigwedge_{x \in \{0,1\}^n} C_x^3.$$

Then we have

$$\text{size}(D) = 2^n (\text{size}(C_x^3)) \text{ and } \text{depth}(D) = \text{depth}(C_x^3) + 1.$$

This completes the construction of D, with

$$\text{size}(D) < 2^{3m^2} \text{ and } \text{depth}(D) \leq 2d + 6 \leq 3d - 1.$$

Below we will denote $3d-1$ by $\widehat{d}$.

From our construction, it follows that (i) the uniform independent distribution on the input bits of D simulates the random restriction ρ with $p = 2^{-L} \approx 1/(20m)^d$, and that (ii) D becomes true if every $C_x|\rho$ has decision tree depth at most $2m$. Hence, the following claim follows from Claim 1.

CLAIM 2.

$$\Pr[D=1] \;\geq\; 1 - \frac{m}{2^{m-n}},$$

where the probability is over uniform input bits of D.

Now we apply an NW generator to this circuit D. First we recall some basic notions on NW generators from [NW94].

Let U, M, m, and q be positive integers. Let $[U]$ be some set of cardinality U, e.g., $\{1,2,\ldots,U\}$. A collection of subsets $\mathcal{S} = \{S_1,\ldots,S_M\}$ of some domain $[U]$ is called an (m,q)-*design* if it satisfies the following conditions.

(1) $\forall i,\ 1 \leq i \leq M,\ [\ |S_i| = q\ ]$, and

(2) $\forall i, \forall j,\ 1 \leq i \neq j \leq M,\ [\ |S_i \cap S_j| \leq m\ ]$.

Based on a given (m,q)-design $\mathcal{S} = \{S_1,\ldots,S_M\}$ with domain $[U]$, we define the following function $g_{\mathcal{S}}$: $\{0,1\}^U \to \{0,1\}^M$, which we call a (parity-based) *NW generator*:

$$\begin{aligned} &g_{\mathcal{S}}(x_1 \cdots x_U) \;=\; y_1 \cdots y_M, \\ &\quad \text{where each } y_i,\ 1 \leq i \leq M, \text{ is defined} \\ &\quad \text{by } y_i \;=\; x_{s_1} \oplus \cdots \oplus x_{s_q} \text{ (where } S_i = \{s_1,\ldots,s_q\} \subseteq [U]). \end{aligned}$$

For the pseudorandomness of this generator, we have the following lemma [NW94].

LEMMA 5. *For any positive integers U, M, m, q, s, and e, and positive real ϵ, let $g_{\mathcal{S}}$ be the NW generator defined using an (m,q)-design $\{S_1,\ldots,S_M\}$ with domain $[U]$, and suppose for any depth $e+1$ circuit C on q input bits and of size at most $s + c_{\mathrm{nw}} 2^m M$ (where c_{nw} is some constant), the q-bit parity function has the following bias:*

$$\left| \Pr_{(u_1,\ldots,u_q)\in\{0,1\}^q}[C(u_1,\ldots,u_q) = u_1 \oplus \cdots \oplus u_q] - \frac{1}{2} \right| \;\leq\; \frac{\epsilon}{M}.$$

Then $g_{\mathcal{S}}$ has the following pseudorandomness against any depth e circuit E on M input bits and of size at most s.

$$\left| \Pr_{\boldsymbol{y}\in\{0,1\}^M}[E(\boldsymbol{y}) = 1] - \Pr_{\boldsymbol{x}\in\{0,1\}^U}[E(g_{\mathcal{S}}(\boldsymbol{x})) = 1] \right| \;\leq\; \epsilon.$$

To apply the NW generator to our depth $\widehat{d}$ circuit D constructed above, we set our parameters and define our (m,q)-design, as follows. For the parameters m and M, we will use the same ones that have been used so far, namely, $m = n^k$ and $M = 2^m$. We will take a finite field $\mathbf{F}$ and set $q = |\mathbf{F}|$ and $U = q^2$. We will take a specific finite field $\mathbf{F} = \mathbf{Z}_2[X]/(X^{2\cdot 3^u} + X^{3^u} + 1)$ [vL91], where each element $\alpha \in \mathbf{F}$ takes $K = 2 \cdot 3^u$ bits, and $q = |\mathbf{F}| = 2^K$. We choose u so that $q \geq (3m^2+1)^{\widehat{d}+2}$. Then $q^{1/(\widehat{d}+2)} \geq \log_2(2^{3m^2} + c_{\mathrm{nw}} 2^m M)$, where c_{nw} is the constant in the above lemma. Clearly $q \leq n^{ckd}$ will do, for some universal constant c, for example, $c = 7$. Then $K = O(dk \log n)$. Thus, this field has polynomial size and each element is represented by $O(\log n)$ bits. All arithmetic operations in this field $\mathbf{F}$ are easy.

We will consider precisely $M = 2^m$ polynomials $f_z(\xi) \in \mathbf{F}[\xi]$, each of degree at most m, where each f_z is indexed by its coefficients, concatenated as a bit sequence

of length exactly m. The precise manner in which this is done is not very important, but for definiteness, we can take the following. We take polynomials of degree $\delta = \lfloor m/K \rfloor = \Omega(n^k/(dk \log n)) \gg n^2$, with exactly $\delta + 1$ coefficients,

$$f_z(\xi) \;=\; c_\delta \xi^\delta + \cdots + c_1 \xi + c_0,$$

where all c_j varies over $\mathbf{F}$, except that c_δ is restricted to exactly $2^{m-K\cdot\delta}$ many values. Note that $0 \le m - K \cdot \delta < K$. The concatenation $z = \langle c_\delta \cdots c_0 \rangle$ has exactly m bits. Each f_z defines a subset of $\mathbf{F} \times \mathbf{F}$ of cardinality q, $\{(\alpha, f_z(\alpha)) \mid \alpha \in \mathbf{F}\}$, which we denote by S_z. An (m, q)-design that we will use is defined as $\mathcal{S} = \{S_1, \ldots, S_M\}$, indexed by $z \in \{0,1\}^m$, which we identify with the index set $\{1, \ldots, M\}$. Note that $\mathbf{F} \times \mathbf{F}$ is a domain $[U]$ with $U = q^2$. The first condition of an (m, q)-design is immediate, and the second condition, i.e., $|S_z \cap S_{z'}| \le m \;\forall z \ne z'$, is also easy to see by noting that $\deg(f_z) < m$ and $\deg(f_{z'}) < m$. Note that our NW generator $g_{\mathcal{S}}$ generates a pseudorandom sequence of length $M = 2^m$ from a seed of length $U = q^2$.

For showing the pseudorandomness of $g_{\mathcal{S}}$, we use the following lemma, which follows from the decision tree version of the Switching Lemma.

LEMMA 6. *For any depth e, and for all sufficiently large q, any circuit C on q inputs and of size at most $2^{q^{1/(e+1)}}$ satisfies*

$$\left| \Pr_{(u_1,\ldots,u_q)\in\{0,1\}^q}[C(u_1, \ldots, u_q) = u_1 \oplus \cdots \oplus u_q] - \frac{1}{2} \right| \;\le\; 2^{-q^{1/(e+1)}}.$$

Then the following claim is immediate from Lemmas 5 and 6.

CLAIM 3. *Our NW generator $g_{\mathcal{S}}$ has the following pseudorandomness against any circuit E of size at most 2^{3m^2} and depth $\widehat{d}$:*

$$\left| \Pr_{\boldsymbol{y}\in\{0,1\}^M}[E(\boldsymbol{y}) = 1] - \Pr_{\boldsymbol{x}\in\{0,1\}^U}[E(g_{\mathcal{S}}(\boldsymbol{x})) = 1] \right| \;\le\; 2^{m-3m^2}.$$

Recall that the circuit D takes $(L+1)M$ Boolean inputs, i.e., $(a_{z,1}, \ldots, a_{z,L}, b_z)$ for $z \in \{0,1\}^m$, where $M = 2^m$ and $L = \lceil \log_2 \frac{1}{p} \rceil$. We provide these input values by our NW generator that produces an M-bit pseudorandom string from a q^2-bit random seed. Hence, for the seed to the generator, a random string of length $(L+1)q^2$ is needed, and we use a sequence of independently and uniformly distributed bits $\{u^{(0)}_{\alpha,\beta}, u^{(1)}_{\alpha,\beta}, \ldots, u^{(L)}_{\alpha,\beta}\}$ for each $\alpha, \beta \in \mathbf{F}$. That is, for each $j = 1, \ldots, L$, we use q^2 bits $\{u^{(j)}_{\alpha,\beta} \mid \alpha, \beta \in \mathbf{F}\}$ to generate the M Boolean values of $a_{z,j}$ for $z \in \{0,1\}^m$. Similarly, the set $\{u^{(0)}_{\alpha,\beta} \mid \alpha, \beta \in \mathbf{F}\}$ of q^2 bits is used to generate the M Boolean values of b_z for $z \in \{0,1\}^m$. More specifically, for each $z \in \{0,1\}^m$ and $j = 1, \ldots, L$, we define $a_{z,j}$ and b_z as follows:

$$a_{z,j} \;=\; \bigoplus_{\alpha\in\mathbf{F}} u^{(j)}_{\alpha,f_z(\alpha)} \;\text{ and }\; b_z \;=\; \bigoplus_{\alpha\in\mathbf{F}} u^{(0)}_{\alpha,f_z(\alpha)}.$$

Then we have the following claim.

CLAIM 4. *Let $\boldsymbol{g}^{(i)}_{\mathcal{S}}$ denote the pseudorandom output sequence of $g_{\mathcal{S}}$ on random seed bits $\{u^{(i)}_{\alpha,\beta} \mid \alpha, \beta \in \mathbf{F}\}$ for $0 \le i \le L$. Then*

$$\Pr[D(\boldsymbol{g}^{(1)}_{\mathcal{S}}, \ldots, \boldsymbol{g}^{(L)}_{\mathcal{S}}, \boldsymbol{g}^{(0)}_{\mathcal{S}}) = 1] \;\ge\; 1 - o(1),$$

where the probability is over independently and uniformly distributed bits $\{u^{(0)}_{\alpha,\beta}, u^{(1)}_{\alpha,\beta}, \ldots, u^{(L)}_{\alpha,\beta}\}$ for $\alpha, \beta \in \mathbf{F}$.

Proof. Let us denote by $\boldsymbol{a}_i$ and $\boldsymbol{b}$, respectively, a sequence of M true random bits assigned to D's input variables $a_{z,i}$ and b_z for $z \in \{0,1\}^m$. Then our goal is to show that $\Pr[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \ldots, \boldsymbol{g}_{\mathcal{S}}^{(L)}, \boldsymbol{g}_{\mathcal{S}}^{(0)}) = 1]$ is close to 1. We claim $\Pr[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \ldots, \boldsymbol{g}_{\mathcal{S}}^{(L)}, \boldsymbol{g}_{\mathcal{S}}^{(0)}) \neq 1] \leq 1/2^{n-1}$. For a contradiction suppose it is $> 1/2^{n-1}$.

Recall that from Claim 2 $\Pr[D(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_L, \boldsymbol{b}) \neq 1] \leq m/2^{m-n}$.

Then we have

$$\left|\Pr[D(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_L, \boldsymbol{b}) \neq 1] - \Pr[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \ldots, \boldsymbol{g}_{\mathcal{S}}^{(L)}, \boldsymbol{g}_{\mathcal{S}}^{(0)}) \neq 1]\right| \;>\; \frac{1}{2^{n-1}} - \frac{m}{2^{m-n}} \;>\; \frac{1}{2^n}.$$

This implies, by the telescoping argument

$$\begin{aligned}
\frac{1}{2^n} &< \left|\Pr[D(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_L, \boldsymbol{b}) \neq 1] - \Pr[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \ldots, \boldsymbol{g}_{\mathcal{S}}^{(L)}, \boldsymbol{g}_{\mathcal{S}}^{(0)}) \neq 1]\right| \\
&\leq \left|\Pr[D(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_L, \boldsymbol{b}) \neq 1] - \Pr[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_L, \boldsymbol{b}) \neq 1]\right| \\
&\quad + \left|\Pr[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_L, \boldsymbol{b}) \neq 1] - \Pr[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \boldsymbol{g}_{\mathcal{S}}^{(2)}, \boldsymbol{a}_3, \ldots, \boldsymbol{a}_L, \boldsymbol{b}) \neq 1]\right| \\
&\quad \cdots \\
&\quad + \left|\Pr[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \ldots, \boldsymbol{g}_{\mathcal{S}}^{(L)}, \boldsymbol{b}) \neq 1] - \Pr[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \ldots, \boldsymbol{g}_{\mathcal{S}}^{(L)}, \boldsymbol{g}_{\mathcal{S}}^{(0)}) \neq 1]\right|,
\end{aligned}$$

that there exists some i such that

$$\begin{aligned}
&\Big| \Pr\left[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \ldots, \boldsymbol{g}_{\mathcal{S}}^{(i-1)}, \boldsymbol{a}_i, \ldots, \boldsymbol{a}_L, \boldsymbol{b}) \neq 1\right] \\
&\quad - \Pr[D(\boldsymbol{g}_{\mathcal{S}}^{(1)}, \ldots, \boldsymbol{g}_{\mathcal{S}}^{(i-1)}, \boldsymbol{g}_{\mathcal{S}}^{(i)}, \boldsymbol{a}_{i+1}, \ldots, \boldsymbol{a}_L, \boldsymbol{b}) \neq 1]\Big| > \frac{1}{L2^n}.
\end{aligned}$$

By an averaging argument, this bound still holds by appropriately fixing random bits other than $\boldsymbol{a}_i$ and the source bits for $\boldsymbol{g}_{\mathcal{S}}^{(i)}$. In other words, for some circuit D' with M input variables of size at most $\text{size}(D) = 2^{3m^2}$ and depth $\text{depth}(D) = \widehat{d}$, we have

$$\left|\Pr[D'(\boldsymbol{a}_i) = 1] - \Pr[D'(\boldsymbol{g}_{\mathcal{S}}^{(i)}) = 1]\right| \;>\; \frac{1}{L2^n}.$$

This is a contradiction of Claim 3, the pseudorandomness of the generator $g_{\mathcal{S}}$, since $L = O(d \log m)$. □

This claim states that with high probability, a pseudorandom sequence satisfies D, meaning that the random restriction induced from the pseudorandom sequence reduces *every* C_x to a simple function (e.g., a small decision tree) whose value can be fixed by fixing $t = 2m$ additional variables (for each C_x) to 0. Next we will argue that for such a pseudorandom restriction, one can find some space to encode the determined value of each C_x.

Consider a restriction induced by a pseudorandom sequence satisfying D. Apply this restriction to all variables z of circuits C_x, and fix further the value of some set Y of variables to 0 in order to determine the value of circuits C_x $\forall x \in \{0,1\}^n$. We may assume that the size of Y is at most $2m2^n$, which is guaranteed by the fact that $D = 1$ with our pseudorandom sequence. Then there exists y_0 of length $n^2/2$ such that a segment $T_{y_0} = \{z \in \{0,1\}^m \mid y_0 \text{ is a prefix of } z\}$ has no intersection with Y; that is, all variables in T_{y_0} are free from any variables used to fix the value of circuits C_x . This is simply because $2m2^n \ll 2^{n^2/2}$. Our plan is to code the results of C_x by a Boolean variable z of the form $z = y_0 x w$ for some w. The key requirements are that

(i) the variable z is assigned $*$ by the pseudorandom restriction, and (ii) it is easy to find such z (i.e., w) from a given x. (We may assume that the string y_0 and the seed for the chosen pseudorandom sequence are remembered by being encoded in the target polynomial-size circuit $C_{\mathcal{M}}$.)

Let $\boldsymbol{u}_{\alpha,\beta}$ be a column vector of 0-1 uniform bits $(u^{(1)}_{\alpha,\beta}, u^{(2)}_{\alpha,\beta}, \ldots, u^{(L)}_{\alpha,\beta})^{\mathrm{T}}$. Recall that in D's simulation of circuits C_x, a Boolean variable z (of C_x) is assigned $*$ iff $a_{z,j} = 0\ \forall j = 1, \ldots, L$. Hence, z is assigned $*$ by a pseudorandom restriction iff $\sum_{\alpha\in\mathbf{F}} \boldsymbol{u}_{\alpha,f_z(\alpha)} = \mathbf{0}$ in $\mathbf{Z}_2^L$. y_0 is determined by the pigeonhole principle and depends on the source bits $\boldsymbol{u}_{\alpha,\beta}$. We also need to have plenty of $*$'s in the segment T_{y_0}. Since we cannot predetermine y_0, we demand all segments T_y have plenty of $*$'s. So, we want our source bits $\boldsymbol{u}_{\alpha,\beta}$ to satisfy the following condition:

$$(1)\quad \forall y \in \{0,1\}^{n^2/2},\ \forall x \in \{0,1\}^n,\ \exists z = yxw \in \{0,1\}^m\ \left[\ \sum_{\alpha\in\mathbf{F}} \boldsymbol{u}_{\alpha,f_z(\alpha)} = \mathbf{0}\ \right].$$

Furthermore, such a w should be easy to compute from the source bits $\boldsymbol{u}_{\alpha,\beta}$, and the given y, and x.

Recall that for any $z \in \{0,1\}^m$, f_z is defined by the sequence of the coefficients $\langle c_\delta \cdots c_0\rangle$, which concatenates to z. Let γ be the largest index such that the binary concatenation $\langle c_\delta \cdots c_\gamma\rangle$ becomes longer than $n^2/2+n$ bits, so $n^2/2+n < |\langle c_\delta \cdots c_\gamma\rangle| \le n^2/2+n+K$. Then for any $y \in \{0,1\}^{n^2/2}$ and $x \in \{0,1\}^n$, we have some subsequence of coefficients $c_\delta, \ldots, c_\gamma$ such that $yx0^v = \langle c_\delta \cdots c_\gamma\rangle$, with some v for padding. Note that $\gamma > 0$, since $m = n^k$ and $k > 2$. We will show (see Claim 5 below) that with high probability a sequence of random source bits $\boldsymbol{u}_{\alpha,\beta}$ satisfies the following:

$$(2)\qquad \forall c_\delta \in \mathbf{F},\ \ldots,\ \forall c_\gamma \in \mathbf{F},\ \exists c_0 \in \mathbf{F}\ \left[\ \sum_{\alpha\in\mathbf{F}} \boldsymbol{u}_{\alpha,f_{z^*}(\alpha)} = \mathbf{0}\ \right],$$

where z^* is a string in $\{0,1\}^m$ that is the concatenation $\langle c_\delta \cdots c_\gamma 0 \cdots 0 c_0\rangle$. Observe that condition (2) is sufficient for our requirement (1). Consider any $y \in \{0,1\}^{n^2/2}$ and $x \in \{0,1\}^n$, and let $c_\delta, \ldots, c_\gamma$ be the coefficients corresponding to $yx0^v$. Then from (2), there exists some c_0 by which we can define $z^* = \langle c_\delta \cdots c_\gamma 0 \cdots 0 c_0\rangle$ satisfying the condition of (1). Furthermore, we will show that we can easily find such c_0 (thus z^*) given $\boldsymbol{u}_{\alpha,\beta}$, and $c_\delta, \ldots, c_\gamma$, by checking all q elements of $\mathbf{F}$.

We now summarize our oracle construction. Choose any setting of the random bits $\omega = \boldsymbol{u}_{\alpha,\beta}$, such that it generates $(L+1)M$ pseudorandom bits Ω satisfying both $D = 1$ and (2); let ρ_Ω be the restriction induced by this pseudorandom sequence. We construct the segment $X^{=m}$ of our oracle by ρ_Ω as follows. Below z denotes a string in $\{0,1\}^m$ whose membership to X has not been determined yet in the construction. Let X_{fixed} (resp., $\overline{X}_{\mathrm{fixed}}$) be the set of strings in $\{0,1\}^m$ whose membership to X (resp., $\overline{X}$) has been determined. Initially, both X_{fixed} and $\overline{X}_{\mathrm{fixed}}$ are empty. First fix the membership according to ρ_Ω; that is, z is put into X_{fixed} (resp., $\overline{X}_{\mathrm{fixed}}$) iff ρ_Ω sets 1 (resp., 0) to the corresponding variable. Secondly, choose a set $Y \subseteq \{0,1\}^m - (X_{\mathrm{fixed}} \cup \overline{X}_{\mathrm{fixed}})$ of at most $2m2^n$ strings such that adding Y to $\overline{X}_{\mathrm{fixed}}$ determines the value of circuits $C_x\ \forall x \in \{0,1\}^n$. This set Y is guaranteed by $D = 1$. Add Y to $\overline{X}_{\mathrm{fixed}}$. Fix one y_0 such that $T_{y_0} \cap Y = \emptyset$. This y_0 exists by the pigeonhole principle. Then for any $x \in \{0,1\}^n$, put any z of the form y_0xw for some w into X_{fixed} (resp., $\overline{X}_{\mathrm{fixed}}$) iff the (already determined) value of C_x is 1 (resp., 0). Then put all remaining z into $\overline{X}_{\mathrm{fixed}}$.

Now we explain how to design a polynomial-size circuit $C_{\mathcal{M}}$ simulating $\mathcal{M}^X$. We may assume that the information on the seed ω (of length $(L+1)q^2 = n^{O(kd)}$) and y_0 are hardwired into the circuit, and they can be used in the computation. For a given input x, the circuit exhaustively searches for $c_0 \in \mathbf{F}$ satisfying the condition of (2) for the coefficients $c_\delta, \ldots, c_\gamma$ corresponding to $y_0 x 0^v$. Since the seed is given, for any $z^* = \langle c_\delta \cdots c_\gamma 0 \cdots 0 c_0 \rangle$, one can compute $\sum_{\alpha \in \mathbf{F}} \boldsymbol{u}_{\alpha, f_{z^*}(\alpha)}$ within polynomial time in n. Also the size of $\mathbf{F}$ is $q = n^{O(kd)}$. Thus, the desired c_0 (and hence z^*) is computable in polynomial time. When z^* is obtained, the circuit queries the oracle whether "$z^* \in X$?" and accepts the input iff $z^* \in X$. It is easy to check whether the whole computation can be implemented by some circuit of size n^{ckd} for some constant $c > 0$.

We complete the proof by proving the following claim.

CLAIM 5. *Over $q^2 L$ independent and uniform random bits $\{u^{(1)}_{\alpha,\beta}, \ldots, u^{(L)}_{\alpha,\beta} \mid \alpha, \beta \in \mathbf{F}\}$, condition (2) holds with probability $1 - o(1)$.*

Proof. For any fixed $c_\delta, \ldots, c_\gamma$, let $z^*(c)$ denote $\langle c_\delta \cdots c_\gamma 0 \cdots 0 c \rangle$. Then $f_{z^*(c)}(\xi)$ is expressed as $f_{z^*(c)}(\xi) = g(\xi) + c$, where the polynomial $g(\xi) = c_\delta \xi^\delta + \cdots + c_\gamma \xi^\gamma$ is independent of c.

Define $\boldsymbol{u}^*_{\alpha,c} = \boldsymbol{u}_{\alpha, g(\alpha)+c}$. Then since $\boldsymbol{u}^*_{\alpha,c} = \boldsymbol{u}_{\alpha, f_{z^*(c)}(\alpha)}$, condition (2) can be stated as

$$\forall c_\delta,\ \ldots,\ \forall c_\gamma,\ \exists c_0 \left[\sum_{\alpha \in \mathbf{F}} \boldsymbol{u}^*_{\alpha,c_0} = \mathbf{0} \right].$$

Notice that for any fixed $c_\delta, \ldots, c_\gamma$, for any α, α', c, and c', the vectors $\boldsymbol{u}^*_{\alpha,c}$ and $\boldsymbol{u}^*_{\alpha',c'}$ consist of disjoint sets of bits, unless $\alpha = \alpha'$ and $c = c'$. Hence, if $c \neq c'$, they are (probabilistically) independent, from which the following bound follows: $\forall c_\delta, \ldots, c_\gamma$,

$$\Pr\left[\forall c_0 \left[\sum_{\alpha \in \mathbf{F}} \boldsymbol{u}^*_{\alpha,c_0} \neq \mathbf{0} \right] \right] = \prod_{c \in \mathbf{F}} \Pr\left[\sum_{\alpha \in \mathbf{F}} \boldsymbol{u}^*_{\alpha,c} \neq \mathbf{0} \right] = \left(1 - \frac{1}{2^L}\right)^q < e^{-\Omega(q/(20m)^d)},$$

where the probability is taken uniformly over all the bits $u^{(1)}_{\alpha,\beta}, \ldots, u^{(L)}_{\alpha,\beta}$ $\forall \alpha, \beta \in \mathbf{F}$. Then the claim is proved as follows:

$$\begin{aligned} &\Pr \left[\forall c_\delta,\ \ldots,\ \forall c_\gamma,\ \exists c_0 \left[\sum_{\alpha \in \mathbf{F}} \boldsymbol{u}^*_{\alpha,c_0}(\alpha) = \mathbf{0} \right] \right] \\ &\quad \geq 1 - 2^{n^2/2+n+K} e^{-\Omega(q/(20m)^d)} = 1 - o(1). \qquad \square \end{aligned}$$

Remark 1. For convenience we assumed in the proof that $k > 2$ and $d \geq 7$. This is only to simplify notation. Clearly $d \geq 7$ is unnecessary. We only need to forgo the estimate of $2d + 6 \leq 3d - 1$ and use $2d + 6$. Also any machine $\mathcal{M}$ in Σ^{p}_d for $d < 7$ can always be considered in a higher level. Similarly, $k > 2$ is not necessary. If one traces through the proof, with slight modification, any real number $k > 1$ is sufficient.

Remark 2. The final computation by the polynomial-size circuit can be done in NC^1. We only need to evaluate some arithmetic operations in the finite field $\mathbf{F}$. It turns out that since elements in $\mathbf{F}$ are represented by $O(\log n)$ bits, the only step that really requires NC^1 is the parity sum of $n^{O(1)}$ terms, when we evaluate the polynomial f_z.

Remark 3. Though the proof is stated for simulating one machine $\mathcal{M}$, it is also possible to construct a single oracle X such that for every d and k, and every Σ^{p}_d

machine $\mathcal{M}$ running in time $O(n^k)$, the language $L(\mathcal{M}^X)$ can be recognized by some polynomial-size circuit family with stringent access to oracle X.

4. Some results on constant depth circuits. As lower bound results on constant depth circuits play a crucial role in this work, we take this opportunity to present some previously unpublished older results of the first author on these circuits. In particular we emphasize the decision tree viewpoint and give some better constants in the exponents than previously published lower bounds. We give a historical account at the end of the section.

The decision tree perspective was first proposed in [Cai86], where a weaker version of the following lemma was proved. The following proof essentially adapts the techniques from [Hås86a].

We say a Boolean function G on variables $\{x_1, \ldots, x_n\}$ is a *t-AND-OR* if $G = G_1 \wedge G_2 \wedge \cdots \wedge G_w$, where each G_i is the OR of at most t literals (a literal is a variable or its complement). Similarly, we say G is a *t-OR-AND* if $G = G_1 \vee G_2 \vee \cdots \vee G_w$, where each G_i is the AND of at most t literals.

As a base step for analyzing general circuits, we first prove the following lemma.

LEMMA 7. *Let G be a t-AND-OR formula $G_1 \wedge G_2 \wedge \cdots \wedge G_w$. Let ρ be a random p-restriction. Then, $\forall \Delta \geq 0$,*

$$\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta\ ] \ \leq\ (5pt)^\Delta. \tag{3}$$

Proof. The lemma is proved by an induction on w. Concerning G_1, immediately there are 2 cases, either $G_1|_\rho \equiv 1$ or $G_1|_\rho \not\equiv 1$. By renaming literals, we may assume $G_1 = \bigvee_{i \in T} x_i$. Then $G_1|_\rho \equiv 1$ is equivalent to $\rho(i) = 1$ for some $i \in T$. If $G_1|_\rho \equiv 1$, we want to prove that the conditional probability that the rest of G has $\mathrm{DC}(G|_\rho) \geq \Delta$ is no larger. If, however, $G_1|_\rho \not\equiv 1$, we want to carefully analyze what happens to the variables in T. All of this will accumulate as some prior condition on ρ. It will be seen that the inductive step will carry a condition that refers to some collection of subsets of variables on each of which ρ has assigned some variable of it in some definite way. In the earlier proof of Yao [Yao85], as well as in the proof of Cai [Cai86], these conditions were explicitly carried along in the proof. The following device used in Håstad's proof [Hås86a] is more elegant.

One makes the stronger claim that for *any* Boolean function F, we have

$$\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta \mid F|_\rho \equiv 1\ ] \ \leq\ \alpha^\Delta, \tag{4}$$

where α will be set to $5pt$, and we agree that the conditional probability is 0 if the condition is not satisfied. The lemma follows from (4) by taking F to be the constant function 1.

The statement (4) is trivially true for $\Delta = 0$, since the right-hand side becomes 1 in this case. Similarly, if $\alpha \geq 1$, then the statement is true. Thus, we may assume $\Delta > 0$ and $\alpha < 1$.

We prove (4) by induction on w. If $w = 0$, then $G \equiv 1$ by definition and the statement holds since the left-hand side is 0. Let $w > 0$. Put $G = G_1 \wedge G'$, where $G' = G_2 \wedge \cdots \wedge G_w$. Now, either $G_1|_\rho \equiv 1$ or $G_1|_\rho \not\equiv 1$. If $G_1|_\rho \equiv 1$, then we have, by induction,

$$\begin{aligned} &\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta \mid F|_\rho \equiv 1, G_1|_\rho \equiv 1\ ] \\ &\quad =\ \Pr[\ \mathrm{DC}(G'|_\rho) \geq \Delta \mid (F \wedge G_1)|_\rho \equiv 1\ ] \ \leq\ \alpha^\Delta. \end{aligned}$$

Now consider the case $G_1|_\rho \not\equiv 1$. We want to prove

$$\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta \mid F|_\rho \equiv 1, G_1|_\rho \not\equiv 1\ ] \ \leq\ \alpha^\Delta \tag{5}$$

as well. We have renamed the variables so that $G_1 = \bigvee_{i\in T} x_i$. Then $G_1|_\rho \not\equiv 1$ means that for each $i \in T$, $\rho(i) = 0$ or $*$. Moreover, since $\Delta > 0$, it cannot be that $\rho(i) = 0\ \forall i \in T$, or else $G_1|_\rho \equiv 0$, and $\mathrm{DC}(G|_\rho) = 0$. Thus, the set of restrictions ρ such that $F|_\rho \equiv 1, G_1|_\rho \not\equiv 1$, and $\mathrm{DC}(G|_\rho) \geq \Delta$ is contained in

$$\bigcup_{\emptyset\neq Y\subseteq T} \{\rho\ :\ \rho(Y) = *, \rho(T-Y) = 0, F|_\rho \equiv 1, \mathrm{DC}(G|_\rho) \geq \Delta\,\}.$$

Suppose $\rho(Y) = *$ and $\rho(T - Y) = 0$ for some $\emptyset \neq Y \subseteq T$.

First we assume $|Y| < \Delta$. Then there must be some assignment $\sigma_Y : Y \to \{0,1\}$, and $\sigma_Y \neq 0^Y$, where we denote by 0^Y the all-0 assignment on Y, such that $\mathrm{DC}(G|_\rho|_{\sigma_Y}) \geq \Delta - |Y|$. For otherwise, one could obtain *some* decision tree of depth $< \Delta$ for $G|_\rho$ by first asking all the variables in Y. Note that such a $\sigma_Y \neq 0^Y$ because the all-0 assignment leads to $G_1|_\rho|_{0^Y} \equiv 0$.

For $\sigma_Y \neq 0^Y$, $G_1|_\rho|_{\sigma_Y} \equiv 1$, so that $G|_\rho|_{\sigma_Y} \equiv G'|_\rho|_{\sigma_Y}$. Then

$$\begin{aligned}&\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta \mid F|_\rho \equiv 1 \wedge \rho(Y) = * \wedge \rho(T-Y) = 0\ ]\\ &\leq \sum_{\substack{\sigma_Y : Y\to\{0,1\}\\ \sigma_Y\neq 0^Y}} \Pr[\ \mathrm{DC}(G'|_\rho|_{\sigma_Y}) \geq \Delta - |Y| \mid F|_\rho \equiv 1 \wedge \rho(Y) = * \wedge \rho(T-Y) = 0)\ ].\end{aligned} \tag{6}$$

Set

$$\begin{aligned} 0^{T-Y} &= \text{the all-0 assignment on } T - Y,\\ \widetilde{F} &= \textstyle\bigwedge_{\tau_Y : Y\to\{0,1\}} F|_{0^{T-Y}}|_{\tau_Y},\ \text{ and}\\ \widetilde{\rho} &= \rho \text{ restricted to the complement of } T;\end{aligned}$$

then under the condition $\rho(Y) = * \wedge \rho(T - Y) = 0$ we have

$$F|_\rho \equiv 1 \iff \widetilde{F}|_{\widetilde{\rho}} \equiv 1.$$

Hence, the sum in (6) has the upper bound

$$\sum_{\substack{\sigma_Y : Y\to\{0,1\}\\ \sigma_Y\neq 0^Y}} \Pr[\ \mathrm{DC}(G'|_{0^{T-Y}}|_{\sigma_Y}|_{\widetilde{\rho}}) \geq \Delta - |Y| \mid \widetilde{F}|_{\widetilde{\rho}} \equiv 1\ ] \ \leq\ (2^{|Y|} - 1)\alpha^{\Delta-|Y|}, \tag{7}$$

by induction.

The upper bound (7) holds for

$$\Pr[\ \mathrm{DC}(G|\rho) \geq \Delta \mid F|\rho \equiv 1 \wedge \rho(Y) = * \wedge \rho(T-Y) = 0\ ] \tag{8}$$

$\forall Y \neq \emptyset$ with $|Y| < \Delta$. However, for $|Y| \geq \Delta$, the bound in (7) holds trivially for a probability (8), since in this case the bound in (7) is ≥ 1, as $|Y| \geq \Delta > 0$ and $\alpha < 1$. Hence in fact it holds $\forall Y \neq \emptyset$.

Let

$$\begin{aligned} a_Y &= \Pr[\ \rho(Y) = * \wedge \rho(T-Y) = 0 \mid F|_\rho \equiv 1 \wedge G_1|_\rho \not\equiv 1\ ],\\ b_Y &= \Pr[\ \rho(Y) = * \mid F|_\rho \equiv 1 \wedge G_1|_\rho \not\equiv 1\ ].\end{aligned}$$

Then

$$b_Y \;=\; \sum_{Y\subseteq Z\subseteq T} a_Z,$$

and by the Möbius inversion formula,

$$a_Y \;=\; \sum_{Y\subseteq Z\subseteq T} (-1)^{|Z-Y|} b_Z.$$

It follows that

$$\begin{aligned}
&\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta \mid F|_\rho \equiv 1, G_1|_\rho \not\equiv 1\ ] \\
&\leq \sum_{\emptyset\neq Y\subseteq T} a_Y \cdot (2^{|Y|}-1)\alpha^{\Delta-|Y|} \\
&= \sum_{Y\subseteq T} a_Y \cdot (2^{|Y|}-1)\alpha^{\Delta-|Y|}.
\end{aligned}$$

Substituting b_Z for a_Y, we have

$$\begin{aligned}
&\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta \mid F|_\rho \equiv 1, G_1|_\rho \not\equiv 1\ ] \\
&\leq \sum_{Y\subseteq T} \sum_{Y\subseteq Z\subseteq T} (-1)^{|Z-Y|} b_Z \cdot (2^{|Y|}-1)\alpha^{\Delta-|Y|} \\
&= \sum_{Z\subseteq T} b_Z \sum_{Y\subseteq Z} (-1)^{|Z-Y|}(2^{|Y|}-1)\alpha^{\Delta-|Y|} \\
&= \sum_{Z\subseteq T} b_Z (-1)^{|Z|}\alpha^\Delta \sum_{Y\subseteq Z} \left[\left(\frac{-2}{\alpha}\right)^{|Y|} - \left(\frac{-1}{\alpha}\right)^{|Y|}\right] \\
&= \alpha^\Delta \sum_{Z\subseteq T} b_Z (-1)^{|Z|} \left[\left(1-\frac{2}{\alpha}\right)^{|Z|} - \left(1-\frac{1}{\alpha}\right)^{|Z|}\right] \\
&= \alpha^\Delta \sum_{Z\subseteq T} b_Z \left[\left(\frac{2}{\alpha}-1\right)^{|Z|} - \left(\frac{1}{\alpha}-1\right)^{|Z|}\right].
\end{aligned}$$

Concerning b_Z, intuitively, under the condition that $F|_\rho \equiv 1 \wedge G_1|_\rho \not\equiv 1$, the probability of $\rho(Z) = *$ is at most $q^{|Z|}$, where $q = p/(p + \frac{1-p}{2}) \approx 2p$, i.e., $b_Z \leq q^{|Z|}$. We already saw that $G_1|_\rho \not\equiv 1$ means that each variable in Z is assigned either 0 or $*$. The additional condition that $F|_\rho \equiv 1$ can only decrease the probability that some variable is assigned an $*$. We will argue this point more carefully. For the moment, we accept the upper bound $b_Z \leq q^{|Z|}$.

Then, since the coefficients of b_Z are nonnegative, we have

$$\begin{aligned}
&\alpha^\Delta \sum_{Z\subseteq T} b_Z \left[\left(\frac{2}{\alpha}-1\right)^{|Z|} - \left(\frac{1}{\alpha}-1\right)^{|Z|}\right] \\
&\leq \alpha^\Delta \sum_{Z\subseteq T} q^{|Z|} \left[\left(\frac{2}{\alpha}-1\right)^{|Z|} - \left(\frac{1}{\alpha}-1\right)^{|Z|}\right] \\
&= \alpha^\Delta \left\{\left[1+q\left(\frac{2}{\alpha}-1\right)\right]^{|T|} - \left[1+q\left(\frac{1}{\alpha}-1\right)\right]^{|T|}\right\}
\end{aligned}$$

$$\leq \alpha^{\Delta} \left\{ \left[1 - q + \frac{2q}{\alpha}\right]^t - \left[1 - q + \frac{q}{\alpha}\right]^t \right\}. \tag{9}$$

At this point, we can recover the bound $(5pt)^{\Delta}$ as follows [Hås86a]. Observe that

$$\left[1 - q + \frac{2q}{\alpha}\right]^t - \left[1 - q + \frac{q}{\alpha}\right]^t \;\leq\; \left(1 + \frac{2q}{\alpha}\right)^t - \left(1 + \frac{q}{\alpha}\right)^t. \tag{10}$$

If we set $c = 1/\log\phi \approx 2.078$, where $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ is the golden ratio, then we have $e^{2/c} - e^{1/c} = 1$. Then, setting $\alpha = cqt < 5pt$, we get

$$\left(1 + \frac{2q}{\alpha}\right)^t - \left(1 + \frac{q}{\alpha}\right)^t \;=\; \left(1 + \frac{2}{ct}\right)^t - \left(1 + \frac{1}{ct}\right)^t \;<\; e^{2/c} - e^{1/c} \;=\; 1.$$

Then

$$\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta \mid F|_\rho \equiv 1, G_1|_\rho \not\equiv 1\ ] \;<\; \alpha^{\Delta}.$$

This completes the proof of

$$\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta \mid F|_\rho \equiv 1\ ] \;<\; (5pt)^{\Delta}. \qquad \square$$

Finally, we show that $b_Z \leq q^{|Z|}$. Note that for $Z \subseteq T$, we have

$$\Pr[\ \rho(Z) = * \mid G_1|_\rho \not\equiv 1\ ] \;=\; q^{|Z|}.$$

This is because $G_1|_\rho \not\equiv 1$ is the same as ρ assigns only $*$ or 0 on T.

We show that $F|_\rho \equiv 1$ cannot increase the probability of $\rho(Z) = *$. This is trivial if $Z = \emptyset$. Suppose $Z \neq \emptyset$. Consider any fixed restriction ρ' on the complement of Z, $\rho' : Z^c \to \{0, 1, *\}$. Then there is a unique extension of ρ' over Z—call it ρ^*—that satisfies $\rho^*(Z) = *$.

We claim that

$$\Pr[\ \rho(Z) = * \mid F|_\rho \equiv 1, G_1|_\rho \not\equiv 1, \rho|_{Z^c} = \rho'\ ] \;\leq\; q^{|Z|}.$$

The event $\rho(Z) = *$ refers to the unique ρ^*, under the condition $\rho|_{Z^c} = \rho'$. If $F|_{\rho^*} \not\equiv 1$, then the above conditional probability is 0 and the claim trivially holds. Otherwise, $F|_\rho \equiv 1$ for all extensions ρ of ρ' to Z. Hence $F|_\rho \equiv 1, G_1|_\rho \not\equiv 1, \rho|_{Z^c} = \rho'$ refers to exactly $2^{|Z|}$ assignments ρ, such that $\rho(i) \in \{0, *\}\ \forall i \in Z$. The claim follows. Lemma 7 is proved. $\square$

If we take $p = \frac{1}{10t}$, then we get the following bound: For any G as in Lemma 7, and $\forall \Delta \geq 0$,

$$\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta\ ] \;\leq\; 2^{-\Delta}. \tag{11}$$

Using this bound as the base case, we can inductively prove Lemma 4.

On the other hand, it is possible to obtain a slightly stronger bound from (9). In fact the use of the inclusion-exclusion formula has been ignored in (10). In the following, we will show this slightly stronger bound.

We will set $q = \beta/t$ for some constant $\beta > 0$, to be determined later. Set

$$\alpha \;=\; \beta \Big/ \ln\left[\frac{1 + \sqrt{1 + 4e^{\beta}}}{2}\right].$$

Then

$$e^{2\beta/\alpha} - e^{\beta/\alpha} \;=\; e^{\beta}.$$

It follows that

$$\begin{aligned}
&\left[1 - q + \frac{2q}{\alpha}\right]^t - \left[1 - q + \frac{q}{\alpha}\right] \\
&= \left[1 + \left(\frac{2\beta}{\alpha} - \beta\right)\frac{1}{t}\right]^t - \left[1 + \left(\frac{\beta}{\alpha} - \beta\right)\frac{1}{t}\right]^t \\
&< e^{\frac{2\beta}{\alpha} - \beta} - e^{\frac{\beta}{\alpha} - \beta} \;=\; 1.
\end{aligned}$$

Replacing the analysis after (9) in the above proof, we obtain the following lemma.

LEMMA 8. *Let G be a t-AND-OR formula $G_1 \wedge G_2 \wedge \cdots \wedge G_w$. For any β, $0 < \beta < t$, let ρ be a random p-restriction, where $p = \frac{\beta}{t-\beta}$, and let $\alpha = \beta / \ln\left[\frac{1+\sqrt{1+4e^{\beta}}}{2}\right]$. Then $\forall \Delta \geq 0$, we have*

$$\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta\ ] \;\leq\; \alpha^{\Delta}.$$

Minimizing α we find at $\beta_0 = 0.227537$, $\alpha_0 = \alpha(\beta_0) \approx 2^{-1.2638031} \approx 0.4164447$. Let $\gamma_0 = \beta_0/2 \approx 0.1137685$. Then we have the following bound. This is a strengthening of (11).

LEMMA 9. *Let G be a t-AND-OR formula $G_1 \wedge G_2 \wedge \cdots \wedge G_w$, and let ρ be a random γ_0/t-restriction. Then $\forall \Delta \geq 0$, we have*

$$\Pr[\ \mathrm{DC}(G|_\rho) \geq \Delta\ ] \;\leq\; \alpha_0^{\Delta}.$$

Proof. Let $q = \beta_0/t$ and $p = \frac{q}{2-q}$. Then $q = \frac{2p}{1+p}$ is the probability of getting a 0 or an $*$ in a random p-restriction.

We have shown that

$$\Pr[\ \mathrm{DC}(G|_{\rho'}) \geq \Delta\ ] \;\leq\; \alpha_0^{\Delta},$$

where ρ' is a random p-restriction.

Since $p > q/2 = \gamma_0/t$, a random γ_0/t-restriction ρ can be realized by first applying a random p-restriction ρ', followed by a $\gamma_0/(pt)$-restriction. Note that if $\mathrm{DC}(G|_{\rho'}) < \Delta$, then $\mathrm{DC}(G|_\rho) < \Delta$. The lemma follows. □

Now consider general constant depth circuits. Denote by $C^d(s,t)$ the class of depth d circuits with bfi[2] $\leq t$, and the number of gates above the first level $\leq s$. Denote by $C^d(s)$ the class of depth d circuits without a bfi condition but with total size $\leq s$. By extending one level with fan-in 1, clearly $C^d(s) = C^{d+1}(s,1)$. (Here in this notation we suppress the number n of variables and the depth d, where s and t are understood to be functions of one or both of them.)

LEMMA 10. *For all $C \in C^d(s, \gamma_0 n^{1/d})$, we have*

$$\Pr[\ \mathrm{DC}(G|_\rho) \geq \gamma_0 n^{1/d}\ ] \;\leq\; s \cdot \alpha_0^{\gamma_0 n^{1/d}} \;\approx\; s \cdot 2^{-0.143781 \cdot n^{1/d}},$$

where ρ is a random $1/n^{\frac{d-1}{d}}$-restriction.

[2] *bfi* is the abbreviation of bottom fan-in, the maximum fan-in of the bottom level gates. By a "bfi condition" we mean a bound of the form bfi $\leq t$ that is given in each context.

Proof. Apply Lemma 9 repeatedly $d-1$ times, each time with a random $1/n^{\frac{1}{d}}$-restriction. Note that any function with decision tree depth $\leq \Delta$ can be expressed both as a Δ-AND-OR as well as a Δ-OR-AND. After switching bottom level AND-OR formulas to OR-AND's, or vice versa, one can merge two successive levels of gates and reduce the depth by 1. Then the lemma follows. □

Let $C \in C^d(s)$ with no bfi requirement. By considering $C \in C^{d+1}(s,1)$ we may first apply Lemma 9 to each of the bottom depth 2 subcircuits with bfi 1, with a random γ_0-restriction. But we can actually do slightly better by looking at it directly.

Fix any 1-OR-AND formula S. (The case with any 1-AND-OR is dual.) S is just a simple OR; by renaming variables, we may assume $S = \bigvee_{i=1}^m x_i$. Fix any $\Delta > 0$. If we apply a random p-restriction ρ, and if ρ assigns any $x_i = 1$, or if ρ assigns all x_i to 0 or $*$ but fewer than Δ of them are assigned $*$, then $\mathrm{DC}(S|_\rho) < \Delta$. Thus

$$\begin{aligned}\Pr[\ \mathrm{DC}(S|_\rho) \geq \Delta\ ] &\leq \sum_{\substack{J\subseteq\{1,\ldots,m\}\\ |J|\geq\Delta}} \Pr[\ \rho(J) = *, \rho(J^c) = 0\ ] \\ &= \left(\frac{1+p}{2}\right)^m \sum_{j=\Delta}^{m} \binom{m}{j} q^j (1-q)^{m-j},\end{aligned}$$

where $\Pr[\ \rho(i) \neq 1\ ] = p + \frac{1-p}{2} = \frac{1+p}{2}$, and $q = \Pr[\ \rho(i) = * \,|\, \rho(i) \neq 1\ ] = \frac{2p}{1+p}$. Hence

$$\Pr[\ \mathrm{DC}(S|_\rho) \geq \Delta\ ] \ \leq\ q^\Delta \left(\frac{1+p}{2}\right)^m \sum_{i=0}^{m} \binom{m}{i} (1-q)^{m-i} \ =\ q^\Delta \ <\ (2p)^\Delta.$$

So if we first apply a random restriction with $p = \frac{\alpha_0}{2} \approx 0.2082223$, with probability $> 1 - s_1 \alpha_0^{\gamma_0 n^{1/d}}$, all bottom level 1-OR-AND subcircuits are switched to $\gamma_0 n^{1/d}$-AND-OR (or all 1-AND-OR switched to $\gamma_0 n^{1/d}$-OR-AND), where s_1 is the total number of level 1 gates in the depth d circuit C, which are the depth 2 gates in the depth $d+1$ circuit with bfi 1. After the switching we get a circuit of depth $d+1$ with bfi $\leq \gamma_0 n^{1/d}$, but with the same type of gates on the 2 levels just above the bottom level gates. After merging these two levels, we get a circuit in $C^d(s', \gamma_0 n^{1/d})$, where $s' = s - s_1$. Now we apply Lemma 10. This gives the following bound.

LEMMA 11. *For all $C \in C^d(s)$, we have*

$$\Pr[\ \mathrm{DC}(G|_\rho) \geq \gamma_0 n^{1/d}\ ] \ <\ s \cdot \alpha_0^{\gamma_0 n^{1/d}} \ \approx\ s \cdot 2^{-0.143781 \cdot n^{1/d}},$$

where ρ is a random $\alpha_0/(2n^{\frac{d-1}{d}})$-restriction.

These results can be used to prove circuit lower bounds for such circuits. Consider any circuit C in $C^d(s, \gamma_0 n^{1/(d-1)})$. Applying $d-2$ rounds of random $1/n^{1/(d-1)}$-restrictions, with probability greater than $1 - s \cdot 2^{-0.143781 \cdot n^{1/(d-1)}}$, we get a circuit in $C^2(1, \gamma_0 n^{1/(d-1)})$ after switching and merging. The number of variables N left has expectation $\mathrm{Exp}[N] = n^{1/(d-1)}$. By Chernoff bound, we have

$$\Pr[\ N \leq \gamma_0 n^{\frac{1}{d-1}}\ ] \ =\ \Pr[\ N - n^{\frac{1}{d-1}} \leq -(1-\gamma_0) n^{\frac{1}{d-1}}\ ] \ <\ e^{-\frac{(1-\gamma_0)^2}{2} \cdot n^{\frac{1}{d-1}}} \ <\ e^{-0.3927 n^{\frac{1}{d-1}}}.$$

Hence, if $s < 2^{0.143781 \cdot n^{1/(d-1)}}$, the probability is approaching 1 that both C is reduced to a circuit in $C^2(1, \gamma_0 n^{1/(d-1)})$ and $N > \gamma_0 n^{1/(d-1)}$. Therefore C does not compute the parity.

LEMMA 12. *For all $C \in C^d(s, \gamma_0 n^{1/(d-1)})$, if C computes the parity function, then its size s must satisfy*

$$s \;\geq\; 2^{0.143781 \cdot n^{1/(d-1)}}.$$

Let $C \in C^d(s)$ with no bfi requirements. As in the proof of Lemma 11 we will separate the bottom level gates from the rest. Thus we first apply a random $\alpha_0/2$-restriction followed by $d-2$ rounds of random $(n^{1/(d-1)})^{-1}$-restrictions. Thus altogether we applied a random $\alpha_0 \cdot (2n^{\frac{d-2}{d-1}})^{-1}$-restriction, and with probability greater than $1 - (s-1) \cdot 2^{-0.143781 \cdot n^{1/(d-1)}}$ we end up with a circuit in $C^2(1, \gamma_0 n^{1/(d-1)})$. By Chernoff bound again, if N is the number of variables left, then $\mathrm{Exp}[N] = \alpha_0 n^{1/(d-1)}/2$, and therefore

$$\Pr[\, N \leq \gamma_0 n^{\frac{1}{d-1}} \,] \;=\; \Pr\left[\, N - \frac{\alpha_0 n^{1/(d-1)}}{2} \leq -\left(\frac{\alpha_0}{2} - \gamma_0\right) n^{\frac{1}{d-1}} \,\right] \;<\; e^{-0.021423 n^{\frac{1}{d-1}}}.$$

Hence, if $s < 2^{0.143781 \cdot n^{1/(d-1)}}$, C does not compute the parity.

THEOREM 13. *For all $C \in C^d(s)$, if C computes the parity function, then its size s must satisfy*

$$s \;\geq\; 2^{0.143781 \cdot n^{1/(d-1)}}.$$

Now we consider the inapproximability-type lower bound. The decision tree depth lower bound is ideally suited for deriving the inapproximability-type lower bound, and the decision tree perspective was introduced precisely for this reason.

Denote for the rest of this section $m = n^{1/d}$. Let C be a depth d circuit. Note that after some restriction ρ, if C is reduced to a decision tree of depth smaller than the number of variables left, then for exactly half of the 0-1 extensions of ρ, C agrees with parity. This is because at every leaf of the decision tree, the circuit C is completely determined. (This property was called *monochromaticity* in [Cai86].)

Consider $\Pr[\, C(x_1, \ldots, x_n) = \oplus(x_1, \ldots, x_n) \,]$, where $\oplus(x_1, \ldots, x_n)$ denotes the parity function, and the probability is over all 2^n assignments. This can be evaluated by first assigning any random restriction, followed by an unbiased 0-1 assignment for all the remaining variables. Let $E = E_1 \wedge E_2$, where E_1 denotes the event that after the random restriction, we end up with a decision tree of depth t, and E_2 denotes the event that the number of variables N assigned to $*$ is more than t, where t will be specified later as $O(m)$. Let $[\, C = \oplus \,]$ denote $[\, C(x_1, \ldots, x_n) = \oplus(x_1, \ldots, x_n) \,]$ for short.

Note first

$$\begin{aligned}\Pr[\, C = \oplus \,] \;&=\; \Pr[E] \cdot \Pr[\, C = \oplus \,|\, E \,] + \Pr[\neg E] \cdot \Pr[\, C = \oplus \,|\, \neg E \,] \\ &=\; \Pr[\, C = \oplus \,|\, E \,] + \Pr[\neg E]\big(\, \Pr[\, C = \oplus \,|\, \neg E \,] - \Pr[\, C = \oplus \,|\, E \,] \,\big).\end{aligned}$$

As we noted $\Pr[\, C = \oplus \,|\, E \,] = \frac{1}{2}$. Then

$$\left| \Pr[\, C = \oplus \,] - \frac{1}{2} \right| \;\leq\; \frac{1}{2} \Pr[\neg E].$$

Also since $\Pr[\, C = \oplus \,] - \Pr[\, C \neq \oplus \,] = 2(\Pr[\, C = \oplus \,] - \frac{1}{2})$, we have

$$|\Pr[\, C = \oplus \,] - \Pr[\, C \neq \oplus \,]| \;\leq\; \Pr[\neg E].$$

Now we specify the parameter of the random restrictions.

First consider any $C \in C^d(s, \gamma_0 m)$. Then let $t = \gamma_0 m$, and we apply Lemma 10. With a random $(n^{\frac{d-1}{d}})^{-1}$-restriction, we have

$$\Pr[\neg E_1] \;\le\; s\alpha_0^t \;\approx\; s \cdot 2^{-0.143781 \cdot m}.$$

Then again by using the Chernoff bound, we estimate $\Pr[\neg E_2] = \Pr[\, N \le \gamma_0 m \,]$ as follows:

$$\Pr[\neg E_2] \;\le\; e^{-\frac{(1-\gamma_0)^2}{2} m} \;<\; e^{-0.3927m}.$$

Thus $\Pr[\neg E_2]$ is dominated by $\Pr[\neg E_1]$. This analysis gives the following bound.

LEMMA 14. *For all $C \in C^d(2^{0.07189 n^{1/d}}, \gamma_0 n^{1/d})$, we have*

$$|\Pr[\, C = \oplus \,] - \Pr[\, C \neq \oplus \,]| \;\le\; 2^{-0.07189 n^{1/d}}.$$

Finally we consider $C \in C^d(s)$ with no bfi condition. This time we have to work harder to optimize the exponents. Our strategy is as follows. We will first assign a $\frac{\alpha_0}{2}$-restriction, and this will give us a depth d circuit *with* bfi $\le \gamma_0 m$. Then we assign $d-2$ rounds of $1/m$-restrictions, each time using Lemma 9 with the same parameters $p = 1/m$ and $t = \Delta = \gamma_0 m$. This will give us a depth 2 circuit *with* bfi $\le \gamma_0 m$. So far the failure probability is $(s-1)\alpha_0^{\gamma_0 m}$. Finally we assign another $1/m$-restriction, but this time using the parameters $t = \gamma_0 m$ and $\Delta = x\gamma_0 m$, where $0 < x < 1$ is to be determined later. The overall failure probability is $< s\alpha_0^{\gamma_0 m} + \alpha_0^{\Delta} + \Pr[\, N \le \Delta \,]$, where N is the number of variables left.

It turns out that if we used the same values for t and Δ for the estimate in the last round, the bound for $\Pr[\, N \le \Delta \,]$ would be too weak. We will use a more exacting form of the Chernoff bound, and then optimize the overall bound by balancing the last two terms with a judicious choice of x.

We use the following version of the Chernoff bound. (See, for example, p. 70 of [MR95].)

$$\Pr[\, N < (1-\delta)\mathrm{Exp}[N] \,] \;<\; \exp\left[-\mathrm{Exp}[N] \cdot (\delta + (1-\delta) \cdot \ln(1-\delta))\right].$$

Here we have

$$\mathrm{Exp}[N] \;=\; \frac{\alpha_0}{2} m \;\text{ and }\; \delta \;=\; 1 - \frac{2\gamma_0 x}{\alpha_0}.$$

We balance the two bounds by setting x such that

$$x\gamma_0 \ln \frac{1}{\alpha_0} \;=\; \frac{\alpha_0}{2} \left[\, \delta + (1-\delta) \cdot \ln(1-\delta) \,\right].$$

This leads to choosing $x = 0.617945$, and we get both

$$\alpha_0^{\Delta} \;<\; 2^{-0.0888488 \cdot m} \;\text{ and }\; \Pr[\, N \le \Delta \,] \;<\; 2^{-0.0888488 \cdot m}.$$

Then by setting $s = 2^{0.07189 \cdot m}$ we get a balanced discrepancy lower bound.

THEOREM 15. *For all $C \in C^d(2^{0.07189 n^{1/d}})$, we have*

$$|\Pr[\, C = \oplus \,] - \Pr[\, C \neq \oplus \,]| \;\le\; 2^{-0.07189 n^{1/d}}.$$

Note that the bound in Theorem 15 is the same as that of Lemma 14 with the bfi condition. Lemma 6 follows from Theorem 15 for large input size.

Remark 4. The original motivation for Furst–Saxe–Sipser [FSS81], where superpolynomial lower bounds were proved for parity against constant depth circuits, was to provide an oracle separation of PH and PSPACE. This was achieved in a breakthrough result by Yao [Yao85], who proved a lower bound of the form $2^{n^{\Omega(1/d)}}$ for parity on n bits for depth d circuits. Yao's bound was further improved by Håstad [Hås86a] from $2^{-n^{\Omega(1/d)}}$ to $2^{-\frac{1}{10}n^{\frac{1}{d-1}}}$, and his proof has become the standard proof. Independently, Yao's work was improved upon in another direction. Cai investigated in [Cai86] whether constant depth circuits of size $2^{n^{\Omega(1/d)}}$ must err on an asymptotically 50% of inputs against parity. This was motivated by another long-standing open problem, that of random oracle separation of PH and PSPACE (see also [Bab87]). To attack this problem, the decision tree point of view was first adopted in [Cai86], although a different but completely synonymous terminology (Master-Player Game and t-monochromaticity) was used. It was proved in [Cai86] that after a suitable random restriction ρ, with high probability, the constant depth circuit $C\mid_\rho$ has decision tree depth smaller than the number of unassigned Boolean variables. In such cases, $\Pr[C = \oplus]$ is exactly $\frac{1}{2}$. Thus the discrepancy

$$|\Pr[C = \oplus] - \Pr[C \neq \oplus]| \tag{12}$$

was shown to be $o(1)$ for circuits of depth d and size $2^{n^{\Omega(1/d)}}$. Implicitly a bound of the form $2^{-n^{\Omega(1/d)}}$ for the discrepancy (12) was proved there as well [Cai86]. The $o(1)$ upper bound for the discrepancy was sufficient for the random oracle separation result, which was the purpose of [Cai86], but one needs Håstad's technique to improve the bound from $2^{-n^{\Omega(1/d)}}$ to $2^{-cn^{\frac{1}{d}}}$ as in Theorem 15. However, the weaker bound $2^{-n^{\Omega(1/d)}}$ would have sufficed for our Theorem 2. Ko [Ko89a] also used circuit lower bounds to establish the following: For any k, one can construct an oracle with which the polynomial hierarchy collapses to exactly k levels.

It was a marvelous application by Nisan and Wigderson [Nis91a, Nis91b, NW88] that turned the inapproximability type of lower bounds based on decision trees on its head and produced an explicit construction—usually considered an upper bound—of a pseudorandom generator provably indistinguishable from true random bits by polynomial-size constant depth circuits. A central ingredient in [Nis91a, Nis91b, NW88] is a suitable combinatorial design. Seen in this way, our proof of Theorem 2 can be viewed as using a *lower bound* (Switching Lemma) to get an *upper bound* (the NW pseudorandom generator) to prove a *lower bound* (to kill all 2^n circuits C_x simultaneously with the pseudorandom assignments) to finally prove an *upper bound* (to be able to code all the computations). And all this is to show that it is impossible to prove a superpolynomial circuit *lower bound* for any fixed language in the Polynomial-time Hierarchy, with a relativizable proof with stringent access to an oracle.

Acknowledgment. We would like to thank the anonymous referees for their detailed comments, which helped us improve the presentation of the paper.

REFERENCES

[Ajt83] M. Ajtai, *Σ^1_1-formulae on finite structures*, Ann. Pure Appl. Logic, 24 (1983), pp. 1–48.

[Bab87] L. Babai, *Random oracles separate PSPACE from the polynomial-time hierarchy*, Inform. Process. Lett., 26 (1987), pp. 51–53.

[BCGKT] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon, *Oracles and queries that are sufficient for exact learning*, J. Comput. System Sci., 52 (1996), pp. 421–433.

[BFT98] H. Buhrman, L. Fortnow, and T. Thierauf, *Nonrelativizing separations*, in Proceedings of the 13th IEEE Conference on Computational Complexity (CCC'98), IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 8–12.

[Cai86] J.-Y. Cai, *With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy*, in Proceedings of the 18th ACM Symposium on Theory of Computing (STOC'86), ACM, New York, 1986, pp. 21–29. See also the journal version in J. Comput. System Sci., 38 (1989), pp. 68–85.

[Cai01] J.-Y. Cai, $S_2^P \subseteq ZPP^{NP}$, in Proceedings of the 42th IEEE Symposium on Foundations of Computer Science (FOCS'01), IEEE Computer Society Press, Los Alamitos, CA, 2001, pp. 620–628.

[CW03] J.-Y. Cai and O. Watanabe, *On proving circuit lower bounds against the polynomial-time hierarchy: Positive and negative results*, in Proceedings of the 9th International Computing and Combinatorics Conference (COCOON'03), Lecture Notes in Comput. Sci. 2697, Springer-Verlag, Berlin, 2003, pp. 202–211.

[CW04] J.-Y. Cai and O. Watanabe, *BPP = PH by polynomially stringent relativization*, Inform. Process. Lett., to appear.

[FSS81] M. Furst, J. Saxe, and M. Sipser, *Parity, circuits, and the polynomial time hierarchy*, in Proceedings of the 22nd IEEE Symposium on Foundations of Computer Science (FOCS'81), IEEE Computer Society Press, Los Alamitos, CA, 1981, pp. 260–270.

[DK00] D.-Z. Du and K.-I Ko, *Theory of Computational Complexity*, John Wiley & Sons, New York, 2000.

[Hås86a] J. Håstad, *Almost optimal lower bounds for small depth circuits*, in Proceedings of the 18th ACM Symposium on Theory of Computing (STOC'86), ACM, New York, 1986, pp. 6–20.

[Hås86b] J. Håstad, *Computational Limitations for Small-Dept Circuits*, MIT Press, Cambridge, MA, 1986.

[He84] H. Heller, *On relativized polynomial and exponential computations*, SIAM J. Comput., 13 (1984), pp. 717–725.

[HPV77] J. E. Hopcroft, W. J. Paul, and L. G. Valiant, *On time versus space*, J. ACM, 24 (1977), pp. 332–337.

[Kan82] R. Kannan, *Circuit-size lower bounds and non-reducibility to sparse sets*, Inform. and Control, 55 (1982), pp. 40–56.

[KL80] R. M. Karp and R. J. Lipton, *Some connections between nonuniform and uniform complexity classes*, in Proceedings of the 12th ACM Symposium on Theory of Computing (STOC'80), ACM, New York, 1980, pp. 302–309.

[Ko89a] K.-I Ko, *Relativized polynomial time hierarchies having exactly k levels*, SIAM J. Comput., 18 (1989), pp. 392–408.

[KW98] J. Köbler and O. Watanabe, *New collapse consequences of* NP *having small circuits*, SIAM J. Comput., 28 (1998), pp. 311–324.

[MVW99] P. B. Miltersen, N. V. Vinodchandran, and O. Watanabe, *Super-polynomial versus half-exponential circuit size in the exponential hierarchy*, in Proceedings of the 5th Annual International Conference on Computing and Combinatorics (COCOON'99), Lecture Notes in Comput. Sci. 1627, Springer-Verlag, Berlin, 1999, pp. 210–220.

[MR95] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[Nis91a] N. Nisan, *Pseudorandom bits for constant depth circuits*, Combinatorica, 11 (1991), pp. 63–70.

[Nis91b] N. Nisan, *Using Hard Problems to Create Pseudorandom Generators*, MIT Press, Cambridge, MA, 1991.

[NW88] N. Nisan and A. Wigderson, *Hardness vs. randomness*, in Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (FOCS'88), IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 2–12.

[NW94] N. Nisan and A. Wigderson, *Hardness vs. randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.

[vL91] J. van Lint, *Introduction to Coding Theory*, 2nd ed., Springer-Verlag, Berlin, 1992.

[Wil83] C. B. Wilson, *Relativized circuit complexity*, in Proceedings of the 24th IEEE Symposium on Foundations of Computer Science (FOCS'83), IEEE Computer Society Press, Los Alamitos, CA, 1983, pp. 329–334.

[Yao85] A. C. Yao, *Separating the polynomial-time hierarchy by oracles*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science (FOCS'85), IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 1–10.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1011–1034

EFFICIENT ALGORITHMS FOR OPTIMAL STREAM MERGING FOR MEDIA-ON-DEMAND*

AMOTZ BAR-NOY† AND RICHARD E. LADNER‡

Abstract. We address the problem of designing optimal off-line algorithms that minimize the required bandwidth for media-on-demand systems that use stream merging. We concentrate on the case where clients can receive two media streams simultaneously and can buffer up to half of a full stream. We construct an $O(nm)$ optimal algorithm for n arbitrary time arrivals of clients, where m is the average number of arrivals in an interval of a stream length. We then show how to adopt our algorithm to be optimal even if clients have a limited size buffer. The complexity remains the same.

We also prove that using stream merging may reduce the required bandwidth by a factor of order $\rho L/\log(\rho L)$ compared to the simple batching solution where L is the length of a stream and $\rho \leq 1$ is the density in time of all the n arrivals. On the other hand, we show that the bandwidth required when clients can receive an unbounded number of streams simultaneously is always at least $1/2$ the bandwidth required when clients are limited to receiving at most two streams.

Key words. media-on-demand, stream merging, dynamic programming, monotonicity property

AMS subject classifications. 68W05, 68W40

DOI. 10.1137/S0097539701389245

1. Introduction.

Media-on-demand is the demand by clients to play back, view, listen to, or read various types of media such as video, audio, and large files with as small as possible startup delays and with no interruptions. The solution of dedicating a private channel to each client for the required media is implausible even with the ever growing available network bandwidth. Thus, multicasting popular media to groups of clients seems to be the ultimate solution to the ever growing demand for media. The first, and most natural, way to exploit the advantage of multicasting is to *batch* clients together. This implies a trade-off between the overall server bandwidth and the guaranteed startup delay. The main advantage of the batching solutions lies in their simplicity. The main disadvantage is that the guaranteed startup delay may be too large.

The pyramid broadcasting paradigm, pioneered by Viswanathan and Imielinski [43, 44], was the first solution that dramatically reduced the bandwidth requirements for servers by using larger receiving bandwidth for clients and by adding buffers to clients. Many papers have followed this line of research; all of them have demonstrated the huge improvement over the traditional batching solutions. We adopt the *stream merging* technique, introduced by Eager, Vernon, and Zahorjan [18, 19]. Stream merging seems to incorporate all the advantages of the pyramid broadcasting paradigm and is very useful in designing and implementing efficient off-line and on-line solutions.

A system with stream merging capabilities is illustrated in Figure 1. The server multicasts the popular media in a staggered way via several channels. Clients may

*Received by the editors May 10, 2001; accepted for publication (in revised form) January 25, 2004; published electronically June 25, 2004.

http://www.siam.org/journals/sicomp/33-5/38924.html

†Computer and Information Science Department, Brooklyn College – CUNY, 2900 Bedford Avenue, Brooklyn, NY 11210 (amotz@sci.brooklyn.cuny.edu). This work was done in part while the author was a member of the AT&T Labs-Research, Shannon Lab, Florham Park, NJ.

‡Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195 (ladner@cs.washington.edu). This work was done in part at AT&T Labs-Research, Shannon Lab, Florham Park, NJ and was partially supported by NSF grants CCR-9732828 and CCR-0098012.

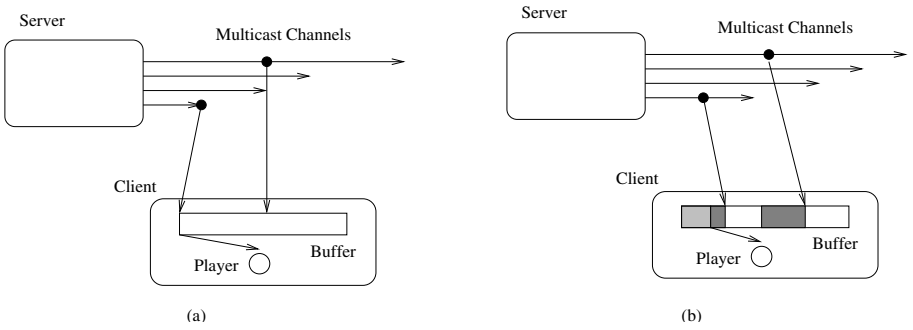


FIG. 1. *The mechanism of receiving data from two streams simultaneously.*

receive data from two streams simultaneously while playing data they have accumulated in their buffers. The playback rate is identical to each of the channels, so that the receiving bandwidth is twice the playback bandwidth. The initial position is illustrated in (a) where the client is about to receive data from a new stream and a stream that was initiated earlier. After some time the system may look as illustrated in (b). The client still receives data from both streams. The top of its buffer, which represents the beginning of the stream, has been viewed by the player. This technique is called stream merging because eventually, as the client receives both the earlier and later streams, it no longer needs the later stream because it already has the data from buffering the earlier one. At this point, if no other client needs the later stream, it can terminate. In a sense the later stream merges with the earlier one, forming just one stream. The termination of the later stream is where bandwidth is saved.

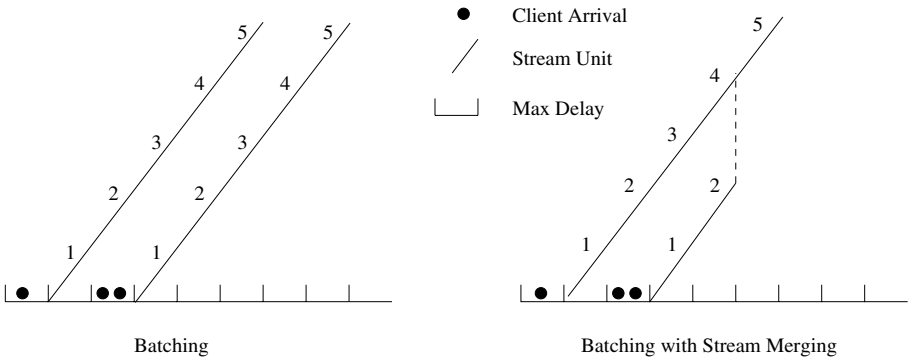


FIG. 2. *The figure on the left shows batching, while the figure on the right shows batching with stream merging.*

It is interesting to contrast stream merging with *batching*, the most common technique for reducing server bandwidth in the presence of multicast. In batching time is divided into intervals. A client that arrives in an interval is satisfied by a full stream at the end of the interval. Bandwidth is saved at the expense of longer guaranteed startup delay for the clients. Stream merging and batching can be combined so that there is a bandwidth saving from both stream merging and batching. Figure 2 shows the difference between pure batching and stream merging with batching. In this figure full streams are of length 5. The three clients require 2 full streams (10 units) with batching alone, but require only 1.4 streams (7 units) with stream merging. The

second and the third clients receive parts 3 and 4 of the first stream at the same time they receive parts 1 and 2 from the second stream; then they receive part 5 from the first stream.

Given a sequence of arrivals, there can be a number of different stream merging solutions to accommodate this sequence. Typically, a stream merging solution has a number of full streams each associated with a number of other truncated streams that eventually merge to this full stream. We measure the bandwidth required by a solution as the sum of the lengths of all the individual (full or truncated) streams in the solutions. We call this sum the *full cost* of the solution. This cost represents the total bandwidth required by the solution and by dividing it by the time span of arrivals it represents the average bandwidth to serve the clients during that time span. In our example of Figure 2, the full cost of the batching solution is 10 units (or 2 streams) and the full cost of the stream merging with the batching solution is 7 units (or 1.4 streams).

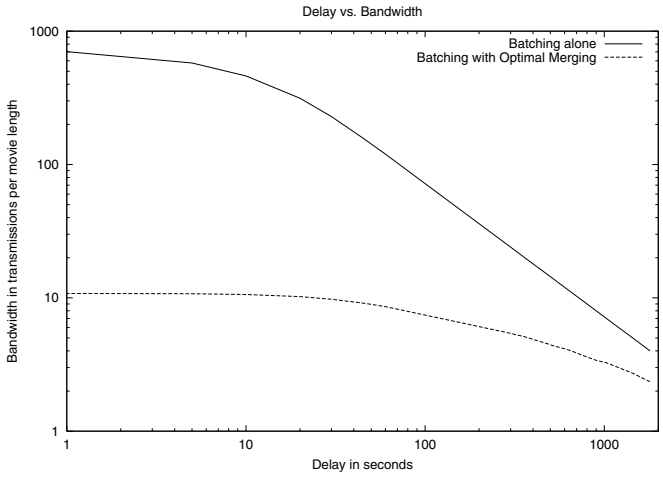


FIG. 3. *Comparison of bandwidth required for batching and batching with optimal stream merging. The figure plots the bandwidth requirement vs. delay for a* 2*-hour movie, with Poisson arrivals averaging every* 10 *seconds.*

Figure 3 shows the bandwidth requirement vs. delay for a popular 2-hour movie, with Poisson arrivals averaging every 10 seconds. The guaranteed startup delay ranges from 1 second to 30 minutes. For stream merging with batching we used an optimal stream merging algorithm. At 1 second delay the difference in bandwidth is dramatic. For batching the bandwidth required is almost the same as it would be if each client had its own stream. On the other hand, at 1 second delay, stream merging with batching uses 1/60 the bandwidth of batching.

1.1. Contributions. The main goal of this paper is to find efficient ways to compute the optimal stream merging solutions, those that minimize the full cost. To determine an optimal solution, we have to decide when to start full streams and how to merge the rest of the streams into the full streams. We assume that the arrival times of clients are known ahead of time and we call this the *off-line* problem, as opposed to the *on-line* problem where client arrivals are not known ahead of time. Computing the optimal off-line solution quickly is a major focus of this paper. The off-line scenario happens when clients make reservations for when their streams will

begin. However, good on-line solutions are required for media-on-demand systems that run in real time. The optimal off-line solution is the gold standard against which on-line solutions should be compared. Fast algorithms for computing an optimal off-line solution allow us to evaluate the quality of on-line solutions on large numbers of media requests.

In our main model a client is capable of receiving two streams simultaneously. We call this the *receive-two* model. It is instructive to consider the *receive-all* model in which a client is capable of receiving any number of streams simultaneously. There are several reasons to consider this case. First, we will see that there is very little gain in going from the receive-two model to the receive-all model. Most of the benefit of stream merging comes from just the ability to receive two streams simultaneously. Second, we will see that many of the results from the receive-two model carry over in a simpler form to the receive-all model.

Our first contribution is a novel model for the stream merging technique. A key concept in our model is that of a *merge tree*, which is an abstraction of the diagram in Figure 2. See Figure 4 for an example. A sequence of merge trees is called a *merge forest*. The root of a merge tree represents a full stream; its structure represents the merging pattern of the remaining streams that are associated with its descendent. A sequence of merge trees is called a *merge forest*. We show that the knowledge of the arrival times and the structure of the merge trees is sufficient to compute the lengths of all the streams and to compute the full cost (total bandwidth) required by the merge forest. A key component of our approach is the concept of *merge cost*. For a given merge tree, the merge cost is the sum of the lengths of all the streams except the root stream. The full cost counts everything, merge cost and the length of the roots for all the merge trees in the forest. This separation into merge cost and full cost helps in designing the optimal algorithms and in having a cleaner analysis. Later in the paper, we first show how to construct an optimal merge tree for a sequence that forms a single tree and then show how to construct the optimal merge forest for a given sequence.

We show several properties that optimal merge trees must have. For example, there is no gain in having streams that do not start at an arrival time of some clients. Other properties will be defined in section 2. These properties were assumed implicitly by all the on-line algorithms that use the stream merging technique [18, 19, 5, 14, 11, 12]. Thus our model, in a way, builds the foundations for designing "good" on-line algorithms.

Our main focus is in designing efficient optimal algorithms in the receive-two model, that is, algorithms that, for a given a sequence of arrivals, either find a merge tree that minimizes merge cost or find a merge forest that minimizes full cost. We have the following results depending on n, the number of arrivals. For the merge cost, we present an efficient $O(n^2)$ time algorithm improving the known $O(n^3)$ time algorithm (see [2, 19]). The latter algorithm is based on a straightforward dynamic programming implementation. Our algorithm implements the dynamic programming utilizing the monotonicity property ([34]) of the recursive definition for the merge cost. For the full cost we use the optimal solution of the merge cost as a subroutine. We describe an $O(nm)$ time algorithm where m is the average number of arrivals in an interval that begins with an arrival and whose length is a full stream length. We also have efficient algorithms for a model in which clients have a limited buffer size. We maintain the $O(nm)$ complexity where this time m is the average number of arrivals in an interval that begins or ends with an arrival and whose length is the minimum

between the stream length and the maximum buffer size.

Additional results establish the performance of the optimal stream merging solutions. Let L be the length of a full stream in slots, where a *slot* is the worst-case waiting time for any arrival before it receives the first segment of the media. For a fixed length media, as the parameter L grows the waiting time tends to zero. Define $\rho \leq 1$ to be the ratio of slots that have at least one arrival to all the slots in a given period of time. We show that an optimal stream merging solution reduces the required full cost by a factor of order $\rho L/\log(\rho L)$ for the full cost compared to the simple batching solution. Note that the improvement is huge for large L because simple batching solutions must dedicate a full stream for each arrival. However, L cannot grow forever because then ρ would approach zero.

Finally, we present optimal algorithms for the receive-all model that have the same time complexity bounds as the receive-two model. We show that the full cost required in an optimal solution in the receive-all model is always at least half the optimal full cost required in the receive-two model.

1.2. Related research. Several papers (e.g., [15, 13, 3]) proposed various batching solutions demonstrating the trade-off between the guaranteed startup delay and the required server bandwidth. The solutions are simple but may cause large startup delays. The seminal pyramid broadcasting solution [43, 44] was the first paper to explore the trade-off with two other resources: the receiving bandwidth of clients and the buffer size of clients. Many researchers were concerned about reducing the buffer size (see e.g., [1]). However, all of them demonstrated the huge improvement over the traditional batching solutions.

The skyscraper broadcasting paper [27] showed that the receive-two model already exploits the dramatic improvement. Researchers also demonstrated the trade-off between the server bandwidth and the receiving bandwidth [26, 20, 37, 38] in this framework. All of these papers assumed a static allocation of bandwidth per transmission. The need for dynamic allocation (or on-line algorithms) motivated the papers [17, 16] that still used the skyscraper broadcasting model. The patching solution [25, 21, 8], the tapping solution [9, 10], the piggybacking solution [2, 23, 24, 35], and the stream merging solution [18, 19] assumed the attractive dynamic allocation of bandwidth to transmissions. The early papers regarding patching assumed that clients may merge only to full streams. Later papers regarding patching assumed a model that is essentially the stream merging model. New research regarding patching [40] assumed that streams may be fragmented into many segments.

The original stream merging algorithms [18, 19] were on-line and event-driven, where telling clients which streams to listen to was done at the time of an event. The specific events were the arrival of a client, the merge time of two streams, and the termination of a stream. The papers reported good practical results compared to the optimal algorithm on Poisson arrivals. These event-driven algorithms are quite different in character from the series of on-line algorithms that appeared subsequently [5, 14, 11, 12]. Unlike in the event-driven algorithms, in the newer algorithms, a client learns all the streams it will be receiving from at the time it arrives. The dynamic Fibonacci tree algorithm of [5] used merge trees and had a competitive analysis. Next, the dyadic algorithm [14] was proposed and analyzed for its average performance on Poisson arrivals. Next, an algorithm based on a new version of merge trees called rectilinear trees was shown to be 5-competitive (full cost no more than 5 times that of the optimal) [11]. Later these same authors proved that the dyadic algorithm is 3-competitive [12]. A comparison of the performance of on-line stream merging

algorithms can be found in [4].

Finally, the following is a partial list of additional papers that address trade-offs among the four parameters: server bandwidth, delay guaranteed, receiving bandwidth, and buffer size: [28, 29, 30, 31, 32, 36, 39, 7, 22, 41, 42].

1.3. Paper organization. In section 2 we define our stream merging model and prove properties of optimal solutions. Section 3 presents our algorithm in the receive-two model with unbounded buffers. In section 4, we consider the limited buffer size case. In section 5, we describe our results for the receive-all model. Finally, we discuss our results and some related problems in section 6.

2. The stream merging model.

Basic definitions. Assume that time starts at 0 and is slotted into unit sized intervals. For example, a 2-hour movie could be slotted into time intervals of 15 minutes. Thus, the movie is 8 units long. For a positive integer t, call the slot that starts at time $t-1$ slot t. The length of a full stream is L units. There are n arrival times for clients denoted by integers $0 \le t_1 < t_2 < \cdots < t_n$. Clients that arrive at the same time slot are considered as one client. At each arrival time a stream must be scheduled, although for a given arrival the stream may not run until conclusion because only an initial segment of the stream is needed by the clients. A client may receive and buffer data from two streams at the same time while viewing the data it accumulated in its buffer. The objective of each client is to receive all the L parts of the stream and to view them without any interruption starting at the time of its arrival.

At this point we would like to note the following:

- We will show later that there is no gain in scheduling streams except at arrival times. Hence, it is very useful to use the client arrival time t as both a name for the client that arrives at time t and for the stream that is initiated at time t. Moreover, for ease of presentation, in the rest of this section we assume that only such streams exist.
- Our results hold for the nondiscrete time model as well by letting the time slots be as small as desired and therefore the value of L as large as needed. We adopt the discrete time model for ease of presentation.

Merge forests and merge trees. A *solution* to an arrival sequence is a *merge forest* which is a sequence of *merge trees*. A merge tree is an ordered labeled tree, where each node is labeled with an arrival time and the stream initiated at that time. The root is labeled t_1 and if a nonroot node is labeled t_i, then its parent is labeled t_j, where $j < i$. This requirement means that a stream can merge only to an earlier stream. Additionally, if t_j is a right sibling of t_i then $j > i$. This requirement means that the children of a node are ordered by their arrival times. Clearly, in a merge forest all the arrival times in one tree must precede the arrival times in the successive tree. We say that an ordered labeled tree has the *preorder traversal property* if a preorder traversal of the tree yields the arrival times in order. Any ordered labeled tree with the preorder traversal property is a merge tree, but not necessarily vice versa. We will see later in Lemma 2.2 that every optimal merge tree satisfies the preorder traversal property.

Figure 4 illustrates a merge tree and a concrete diagram showing how merging would proceed for the given merge tree. In the concrete diagram each arrival is shown on the time axis and for each arrival a new stream is initiated. The vertical axis shows the particular unit of the stream that is transmitted. The root stream, t_1, is of full length, while all the other streams are truncated. A stream is truncated because all

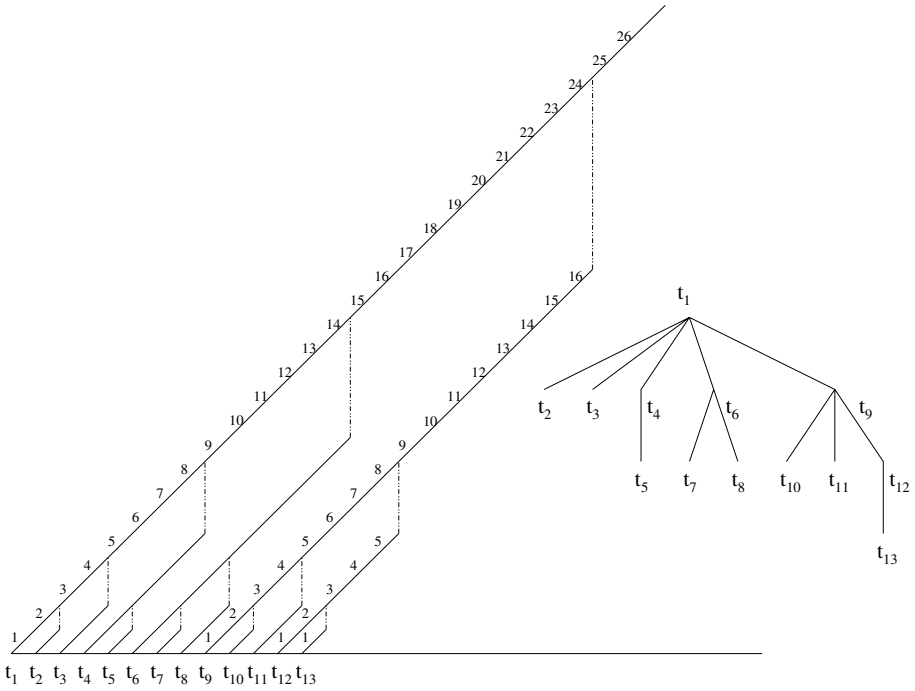


FIG. 4. *On the left is a concrete diagram showing the length of each stream with its merging pattern. On the right is its corresponding merge tree. In this example there are* 13 *arrivals at times* $0, \ldots, 12$.

the clients that were receiving the stream no longer need any data from it, having already received the data from some other stream(s). Note that although the merge tree does not show the stream lengths, it implicitly contains all the information in the concrete diagram, as will be shown in Lemma 2.1.

For the moment, we postpone the explanation of how we calculate the lengths of the truncated streams because we need more explanations of how merging works. Nonetheless, we can now explain that the problem we are addressing is how to find a solution that minimizes the sum of the lengths of all the streams in the solution. This is equivalent to minimizing the total number of units (total bandwidth) needed to serve all the clients. Minimizing the total bandwidth is essentially the same as minimizing the average bandwidth needed to satisfy the requests. The average bandwidth required to satisfy the requests by the forest F is the sum of the total bandwidth required by F divided by $(t_n - t_1)$ which is the time span of the n arrivals. The equivalence follows since the quantity $t_n - t_1$ is independent of the solution.

Receiving procedures. Clients receive and buffer data from various streams according to their location in the forest. At any one time a client can receive data from at most two streams. Informally, a client arriving at time x receives data from all the nodes on the path from x to the *root* of the tree. At the same time it receives data from a node y and its parent until it does not need any more data from the node. At that point the client moves closer to the root by receiving data from the parent of y and its parent. We call this transition a merge operation. In the following we formally define the actions of a client in the merge tree.

Let $x_0 < x_1 < \ldots < x_k$ be the path from the root x_0 to node x_k that is the

arrival time of a specific client. We call this sequence of length $k+1$ the *receiving procedure* of the client. Denote by $x_0, x_1, \ldots, x_k$ the streams that are scheduled at the corresponding arrival times. The client obeys the following *stream merging rules*.

Stage i, $0 \le i \le k-1$: For $x_{k-i} - x_{k-i-1}$ time slots from time $2x_k - x_{k-i}$ to time $2x_k - x_{k-i-1}$ the client receives parts $2x_k - 2x_{k-i} + 1, \ldots, 2x_k - x_{k-i} - x_{k-i-1}$ from stream x_{k-i} and parts $2x_k - x_{k-i} - x_{k-i-1} + 1, \ldots, 2x_k - 2x_{k-i-1}$ from stream x_{k-i-1}.

Stage k: For $L - 2(x_k - x_0)$ time slots from time $2x_k - x_0$ to time $x_0 + L$ the client receives parts $2(x_k - x_0) + 1, \ldots, L$ from stream x_0.

This describes how the client arriving at x_k receives the entire transmission of the stream. In particular, part j of the stream is received in stage $i = k$ if $2(x_k - x_0) < j$ and in stage $i < k$ if $2(x_k - x_{k-i}) < j \le 2(x_k - x_{k-i-1})$. Notice that if $x_k - x_0 \le \lfloor L/2 \rfloor$, then the client is busy receiving data for $L - (x_k - x_0)$ time slots since in $x_k - x_0$ slots it receives data from two streams, and if $x_k - x_0 > \lfloor L/2 \rfloor$, then the client is busy receiving data for $x_k - x_0$ time slots since in $L - (x_k - x_0)$ slots it receives data from two streams.

Consider the example depicted in Figure 4. Assume a full stream of length 26 and the following stream merging rules for the client that arrives at time $t_{13} = 12$. In this case, we have $k = 3$ with $x_0 = 0, x_1 = 8, x_2 = 11, x_3 = 12$. From time 12 to time 13 the client receives part 1 from stream x_3 and part 2 from stream x_2. From time 13 to time 16 the client receives parts $3, \ldots, 5$ from stream x_2 and parts $6, \ldots, 8$ from stream x_1. From time 16 to time 24 the client receives parts $9, \ldots, 16$ from stream x_1 and parts $17, \ldots, 24$ from stream x_0. Finally, from time 24 to time 26 the client receives parts $25, 26$ from stream x_0.

Length of streams. Given the stream merging rules, we must still determine the minimum length of each stream so that all the clients requiring the stream receive their data. In a merge tree T the root is denoted by $r(T)$. If x is a node in the merge tree, then we define $\ell_T(x)$ to be its *length* in T. That is, $\ell_T(x)$ is the minimum length needed to guarantee that all the clients can receive their data from stream x using the stream merging rules. For a nonroot node x define $p_T(x)$ to be its parent and $z_T(x)$ to be the latest arrival time of a stream in the subtree rooted at x. If x is a leaf, then $z_T(x) = x$. We drop the subscript T when there is no ambiguity.

We can see from our definition of the stages that the length L of the root stream must satisfy $z - r(T) \le L - 1$, where z is the last arrival in the merge tree T. Otherwise, the clients arriving at z do not receive data from the stream initiated at $r(T)$. The next lemma shows how to compute the lengths of all the nonroot streams.

LEMMA 2.1. *Let $x \ne r(T)$ be a nonroot node in a tree T. Then*

$$\ell(x) = 2z(x) - x - p(x). \tag{1}$$

In particular, if x is a leaf, then $\ell(x) = x - p(x)$ since $z(x) = x$.

Proof. First observe that if clients $y' < y$ both receive data from x, then client y receives later parts of the stream x. This implies that the length of the stream x is dictated by the needs of the client that arrives at time $z(x)$. Let $x_0, x_1, \ldots, x_k$ be the path from the root of the tree T that contains both x and $z(x)$. That is, $x = x_i$ and $p(x) = x_{i-1}$ for some $i > 0$ and $z(x) = x_k$. By the stream merging rule of stage $k - i$, the client $z(x)$ receives data from the stream $x = x_i$ until time $2x_k - x_{i-1} = 2z(x) - p(x)$. Since $z(x)$ is the last client requiring stream x, then no more transmission of stream x is required. Since the stream x begins at time x and ends at time $2z(x) - p(x)$, its length is $2z(x) - x - p(x)$. ∎

In this paper, we will use for $\ell(x)$ either expression (1) or the following two alternative expressions:

$$\ell(x) = (x - p(x)) + 2(z(x) - x) \tag{2}$$

$$= (z(x) - x) + (z(x) - p(x)). \tag{3}$$

Expression (3) could be viewed as follows. The length of the stream x is composed of two components. The first component is the time needed for clients arriving at time x to receive data from stream x before they can merge with stream $p(x)$. The second component is the time stream x must spend until the clients arriving at time $z(x)$ merge to $p(x)$.

2.1. The merge cost. Let T be a merge tree. The *merge cost* of T is defined as

$$\text{Mcost}(T) = \sum_{x \neq r(T) \in T} \ell(x).$$

That is, the merge cost of a tree is the sum of all lengths in the tree except the length of the root of the tree. For an arrival sequence $t_1, \ldots, t_n$, define the *optimal merge cost* for the sequence to be the minimum cost of any merge tree for the sequence. An *optimal merge tree* is one that has optimal merge cost. The following technical lemma justifies restricting our attention to merge trees with the preorder traversal property.

LEMMA 2.2. *Every optimal merge tree satisfies the preorder traversal property.*

Proof. The proof is by induction on the number of arrivals. The lemma is clearly true for one arrival. Assume we have $n > 1$ arrivals and the lemma holds for any number of arrivals less than n. Let T be an optimal merge tree for the arrivals and let x be the last arrival to merge to the root r of T. Define T_R to be the subtree of T rooted at x and let T_L be the subtree of T obtained by removing T_R. By the induction hypothesis we can assume that T_R and T_L both have the preorder traversal property. Let w be the last arrival in the subtree T_L and let z be the last arrival in the subtree T_R. If $w < x$, then the entire tree T must already have the preorder property, and we are done. We need consider only the case where $w > x$. In this case we will construct another merge tree T' for the same arrivals whose merge cost is less than T's, contradicting the optimality of T.

Define a *high tree* to be a subtree of T_L whose root is greater than x and whose parent of the root is less than x. Let T' be the tree T where all the high trees are removed from T_L and are inserted as children of x. Naturally, the high trees must be inserted so that all the children of x in T' are in arrival order. For all nodes u in T such that $u \neq x$ or u is not an ancestor of a root of a high tree, we have $\ell_T(u) = \ell_{T'}(u)$. For an ancestor u of a root of a high tree we have $\ell_T(u) > \ell_{T'}(u)$, and for x we have $\ell_T(x) < \ell_{T'}(x)$. Let p be the parent of the root of the high tree containing w. We must have $p \neq r$ for otherwise x would not be the last arrival to merge to the root because the root of the high tree containing w is greater than x. We can just examine the change in length of the nodes p and x. We have

$$\text{Mcost}(T) - \text{Mcost}(T') \geq \ell_T(p) - \ell_{T'}(p) + \ell_T(x) - \ell_{T'}(x).$$

Let w' be the largest arrival in the tree rooted at p in T'. We must have $w' < x$; otherwise, w' is in some high tree that was removed from T_L and made a child of x in T'. Since w is the largest arrival in the tree rooted at p in T, we have $\ell_T(p) - \ell_{T'}(p) =$

$2(w - w')$ by Lemma 2.1. There are two cases to consider depending on whether $w < z$ or $w > z$. If $w < z$, then $\ell_T(x) = \ell_{T'}(x)$ because z is the largest arrival in the subtree rooted at x in both T and T'. By definition $w > w'$; hence,

$$\text{Mcost}(T) - \text{Mcost}(T') \geq 2(w - w') > 0.$$

If $w > z$, then $\ell_T(x) - \ell_{T'}(x) = 2(z - w)$ by Lemma 2.1. Hence,

$$\text{Mcost}(T) - \text{Mcost}(T') \geq 2(w - w') + 2(z - w) = 2(z - w') > 0$$

because $w' < x \leq z$. □

Lemma 2.2 allows us to consider only merge trees with the preorder traversal property. As a consequence, henceforth, *we assume that all merge trees have the preorder traversal property.* Hence, a key property of merge trees is that for any node t_i, the subtree rooted at t_i contains the interval of arrivals $t_i, t_{i+1}, \ldots, t_j$, where $z(t_i) = t_j$. Furthermore, t_j is the rightmost descendant of t_i. As a result, we can recursively decompose any merge tree into two in a natural way as shown in the following lemma and seen in Figure 5.

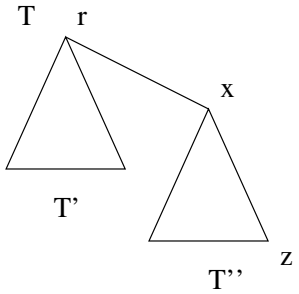


FIG. 5. *The recursive structure of a merge tree T with root r. The last arrival to merge directly with r is x. All the arrivals before x are in T' and all the arrivals after x are in T'' and z is the last arrival.*

LEMMA 2.3. *Let T be a merge tree with root r and last stream z and let x be the last stream to merge to the root of T.*

$$\text{Mcost}(T) = \text{Mcost}(T') + \text{Mcost}(T'') + 2z - x - r, \tag{4}$$

where T' is the subtree of all arrivals before x including r and T'' is the subtree of all arrivals after and including x.

Proof. The length of any node in T' and T'' is the same as its length in T. Since the root of T' is the root of T, it follows that x is the only node in $\text{Mcost}(T)$ whose length is not included in $\text{Mcost}(T')$ or $\text{Mcost}(T'')$. The lemma follows, since by Lemma 2.1 the length of x is $2z(x) - x - p(x) = 2z - x - r$. □

We now prove that there is no gain in broadcasting a prefix of the full stream if there is no arrival for it. That is, an optimal merge tree does not contain a node that represents a prefix of the stream if this prefix does not start at t_i for some $1 \leq i \leq n$. We first prove a lemma that shows that adding nodes to a merge tree always increases the merge cost.

LEMMA 2.4. *Let T be a merge tree and let $x \in T$ be one of its nodes. Then there exists a merge tree T' on the nodes $T - \{x\}$ such that* $\text{Mcost}(T') < \text{Mcost}(T)$.

Proof. Assume first that x is a leaf. Then let T' be T without x. We get that $\text{Mcost}(T) \geq \text{Mcost}(T') + \ell_T(x)$ and therefore $\text{Mcost}(T') < \text{Mcost}(T)$. Let x be a nonleaf node of T and let w be its leftmost child in T. The first modification is for node w. If x is not the root of T, then let the parent of x in T be the parent of w in T'. Otherwise, make w the root of T'. The second modification is for the rest of the children of x in T. Make all of them children of w in T' and add them after w's own children, preserving their original order in T. The rest of the nodes maintain in T' their parent-child relationship from T. By (1),

$$\ell_T(v) = \ell_{T'}(v) \quad \text{for } p_T(v) \neq x. \tag{5}$$

That is, the length of any node v that is not a child of x in T remains the same in T' because there is no change in $p(v)$ and $z(v)$. If $v \neq w$ is a child of x, then (1) implies that

$$\ell_T(v) - \ell_{T'}(v) = w - x > 0 \quad \text{for } v \neq w \text{ and } p_T(v) = x, \tag{6}$$

since w is a later arrival than x. As for w, there are two cases to consider depending on whether w is the root of T' or not. If w is the root of T', then x is the root of T. Hence, $\ell_T(x)$ is not counted in $\text{Mcost}(T)$ and $\ell_{T'}(w)$ is not counted in $\text{Mcost}(T')$. Hence, we have by inequalities (5) and (6)

$$\text{Mcost}(T) - \text{Mcost}(T') = \ell_T(w) + \sum_{(v \neq w) \wedge (p_T(v) = x)} (\ell_T(v) - \ell_{T'}(v)) > 0.$$

If w is not the root of T', then we might have $\ell_{T'}(w) > \ell_T(w)$. However, this is more than compensated for by the inequality

$$\ell_T(x) > \ell_{T'}(w). \tag{7}$$

To see inequality (7), note that $z_T(x) = z_{T'}(w)$ and that $p_T(x) = p_{T'}(w)$; hence by (1), $\ell_T(x) - \ell_{T'}(w) = w - x > 0$. By combining inequalities (5), (6), and (7) we obtain the following:

$$\text{Mcost}(T) - \text{Mcost}(T') = \ell_T(x) + \ell_T(w) - \ell_{T'}(w) + \sum_{(v \neq w) \wedge (p_T(v) = x)} (\ell_T(v) - \ell_{T'}(v)) > 0,$$

which is our desired result. ∎

LEMMA 2.5. *For arrivals $t_1, t_2, \ldots, t_n$ every node (stream) x in an optimal merge tree starts at time t_i for some $1 \leq i \leq n$.*

Proof. Assume to the contrary that there exists a stream x that starts at time t that is not one of the n arrival times $t_1, \ldots, t_n$. By definition, no client needs stream x. Hence, by Lemma 2.4, we could omit node x from T to get a tree T' without x, whose merge cost is smaller than the merge cost of T and is a contradiction to the optimality of T. ∎

Remark. Optimal merge trees also give a lower bound on the bandwidth for the more dynamic event-driven algorithms [18, 19]. A client's receiving procedure, which streams it listens to and when, is determined, in part, by future arrivals. Nonetheless, in the end, the final receiving pattern of a client forms a path in a merge tree. At any point in time only a set of subtrees of the final merge tree is known. Each root of a subtree represents an active stream at that time. When a merge event occurs, the root of some subtree becomes the child of some root in another subtree.

2.2. The full cost. Let F be composed of s merge trees $T_1, \ldots, T_s$. The *full cost* of F is defined as

$$\text{Fcost}(F) = s \cdot L + \sum_{1 \le i \le s} \text{Mcost}(T_i).$$

The above definition is a bit problematic since in the way we define the merge cost it could be the case that a length of a stream is L or larger. Consider the following example. Suppose that the root arrives at time 0 and there are two additional arrivals at times $L-2$ and $L-1$. In one optimal merge tree the third arrival first merges with the second arrival and then both merge with the root; that is, $p(L-1) = L-2$ and $p(L-2) = 0$. The cost of this tree is L for the root, L for the second arrival, and 1 for the third arrival for a total cost of $2L+1$. It is clear that this single merge tree can be considered as a merge forest of two merge trees, the first with one arrival, 0, and the second with two, $L-2$ and $L-1$. A more serious problem is exposed by the following example where the arrival times are 0, $L-3$, and $L-1$. In this case the definition of a merge tree would allow $p(L-1) = L-3$ and $p(L-3) = 0$. In this case the full cost of the merge tree is $2L+3$. Length L for the root 0, length $L+1$ for $L-3$, and length 2 for $L-1$. We have $\ell(L-3)$ greater than L. However, this merge tree is not an optimal merge forest for these three arrivals. The optimal merge forest has two trees, one with root 0 and one with root $L-3$ for a full cost of $2L+2$.

Naturally, we cannot allow the length of any stream to be greater than L. To remedy this problem we define an *L-tree* to be a merge tree in which the length of each stream has length less than or equal to L and the length of the root is L. The first example above is an L-tree, but the second is not. It should be clear that an L-tree with a nonroot x of length L can be split into two L-trees of the same cost by simply making x a new root. An *L-forest* is a merge forest that is composed of L-trees only. For an arrival sequence $t_1, \ldots, t_n$ and stream length L define the *optimal full cost* for the sequence to be the minimum full cost of any L-forest for the sequence. An *optimal L-forest* is one that has optimal full cost.

Our strategy for searching for the optimal L-forest is to consider all possible merge forests as candidates for the optimal. The following lemma shows that this extended search always yields an L-forest as the optimal.

LEMMA 2.6. *Any merge forest F that minimizes* $\text{Fcost}(F)$ *is an L-forest.*

Proof. Define the following split operation on trees. Let T be a merge tree on the arrivals $t_1, \ldots, t_n$. Let $x = t_i$ be a node in the tree. Then the x-split of T creates two trees: T' and T''. T' is rooted at t_1 and contains the arrivals $t_1, \ldots, t_{i-1}$ with the same parent-child relation as in T. T'' is rooted at x and contains the arrivals $t_i, \ldots, t_n$. The parent relation in T'' is defined as follows: Let $y = t_j$ for $i < j \le n$ and let $w = p(y)$ be the parent of y in T. If $w > x$, then $w = p(y)$ in T'' as well. Otherwise, $x = p(y)$ in T''.

Let T be a non-L-tree and let $x \in T$ be a node whose length is $\ell(x) > L$. We claim that

$$\text{Fcost}(T') + \text{Fcost}(T'') < \text{Fcost}(T). \tag{8}$$

We prove this claim by showing that the length of each node, other than x, in T' or T'' is no more than its length in T. Since the length of x is greater than L in T and equal to L in T'', we are done. There are two cases to consider: (i) The length of all the nodes that have the same parent in T' or T'' as they had in T remains the same;

(ii) By Lemma 2.1, the length of all nodes y such that $w = p(y)$ in T but $x = p(y)$ in T'' is reduced since $w < x$.

To prove the lemma, let F be a merge forest that minimizes $\mathrm{Fcost}(F)$. If F is not an L-forest, then there is some merge tree T in F which is not an L-tree. Apply the above split procedure to this tree to obtain a new merge forest with less cost than F. Thus, F must be an L-forest. □

3. The optimal algorithm. In this section we give efficient algorithms for finding a merge tree that minimizes the merge cost and for finding a merge forest that minimizes the full cost. For the merge cost case we assume that the root has length infinity and that all the arrivals can merge to it. In the full cost case we assume that the length of a full stream is L. We then search for the best assignment of roots among the n arrivals. Although some of the assignments may lead to non-L-trees (trees in which some of the nodes have length greater than L), by Lemma 2.6 we know that an optimal merge forest is an L-forest.

For the merge cost we present an efficient $O(n^2)$ time algorithm improving the known $O(n^3)$ time algorithm (see [2, 19]). The latter algorithm is based on a straightforward dynamic programming implementation. Our algorithm implements the dynamic programming utilizing the monotonicity property of the recursive definition for the merge cost. For the full cost we use the optimal solution of the merge cost as a subroutine. We describe an $O(nm)$ time algorithm where m is the average number of arrivals in an interval of length $L - 1$ that begins with an arrival.

3.1. Optimal merge cost. Let $t_1, t_2, \ldots, t_n$ be a sequence of arrivals. Define $M(i, j)$ to be the optimal merge cost for the input sequence $t_i, \ldots, t_j$. In a dynamic programming fashion we show how to compute $M(i, j)$. The optimal cost for the entire sequence is $M(1, n)$. By Lemma 2.3 we can recursively define

$$M(i,j) = \min_{i<k\le j} \{M(i,k-1) + M(k,j) + (2t_j - t_k - t_i)\} \tag{9}$$

with the initialization $M(i, i) = 0$. Using the notation of Lemma 2.3, t_i is the root r, t_j is the last arrival z, and we are looking for the optimal last arrival t_k, which is x, that merges to the root. This recursive formulation naturally leads to an $O(n^3)$ time algorithm using dynamic programming. The following theorem shows that this can be significantly improved.

THEOREM 3.1. *An optimal merge tree can be computed in time $O(n^2)$.*

Proof. To reduce the time to compute the optimal merge cost to $O(n^2)$ we employ monotonicity, a classic technique pioneered by Knuth [33, 34]. Define $r(i, i) = i$ and for $i < j$

$$r(i,j) = \max\{k : M(i,j) = M(i,k-1) + M(k,j) + 2t_j - t_k - t_i\}.$$

That is, $r(i, j)$ is the last arrival that can merge to the root in some optimal merge tree for $t_i, \ldots, t_j$. *Monotonicity* is the property that for $1 \le i < n$ and $1 < j \le n$

$$r(i,j-1) \le r(i,j) \le r(i+1,j). \tag{10}$$

We should note that there is nothing special about using the max in the definition of $r(i, j)$; the min would yield the same inequality (10). Once monotonicity is demonstrated then the search for the k in (9) can be reduced to $r(i + 1, j) - r(i, j - 1) + 1$ possibilities from $j - i$ possibilities. Hence, the sum of the lengths of all the search

intervals is reduced to $\sum_{1\le i<n}\sum_{i<j\le n}(r(i+1,j)-r(i,j-1)+1)=O(n^2)$ from $\sum_{1\le i<j\le n}(j-i)=O(n^3)$. This yields an $O(n^2)$ algorithm.

Fortunately, for our problem we can apply the very elegant method of quadrangle inequalities, pioneered by Yao [45] and extended by Borchers and Gupta [6], that leads to a proof of monotonicity. Define $h(i,k,j)=2t_j-t_k-t_i$ which is the third term in (9). Borchers and Gupta show that if h satisfies the following two properties, then monotonicity holds. For $i\le j<t\le k\le l$ and $i<s\le l$,

1. if $t\le s$, then $h(i,t,k)-h(j,t,k)+h(j,s,l)-h(i,s,l)\le 0$ and $h(j,s,l)-h(i,s,l)\le 0$;
2. if $s\le t$, then $h(j,t,l)-h(j,t,k)+h(i,s,k)-h(i,s,l)\le 0$ and $h(i,s,k)-h(i,s,l)\le 0$.

In our case, both four-term sums are identically zero, while $h(j,s,l)-h(i,s,l)=t_i-t_j\le 0$ and $h(i,s,k)-h(i,s,l)=2(t_k-t_l)\le 0$.

As a byproduct of the computation of $r(i,j)$ we can recursively compute the optimal merge tree using the recursive characterization of Lemma 2.3. We define a recursive procedure for computing an optimal merge tree for the input $t_i,\ldots,t_j$ as follows. If $i=j$, then return the tree with one node labeled t_i. Otherwise, recursively compute optimal merge trees T' for the input $t_i,\ldots,t_{r(i,j)-1}$ and T'' for $t_{r(i,j)},\ldots,t_j$, then attach the root of T'' as an additional last child of the root of T' and return the resulting tree. This procedure is then called for the input $t_1,\ldots,t_n$ to get the final result. With an elementary data structure, and with $r(i,j)$ already computed for $1\le i\le j\le n$, the construction of the optimal merge tree can be done in linear time. □

We conclude this subsection with an upper bound on the merge cost of an arrival sequence $t_1,t_2,\ldots,t_n$. Denote by $N=t_n-t_i$ the span of the arrivals. We are looking for an upper bound that depends only on N and n and not on the sequence itself. In the following theorem we establish an $O(N\log n)$ upper bound based on a full binary merge tree.

THEOREM 3.2. *The optimal merge cost is $O(N\log n)$.*

Proof. Using the notation of this subsection, we prove that

$$M(i,j)\le c(t_j-t_i)\log_2(j-i+1)$$

by induction on $h=j-i$, for some constant $c\ge 4$. For the rest of the proof we omit the base 2 from the log function. The theorem follows by choosing $i=1$ and $j=n$. The claim trivially holds for $h=0$. For $h=1$ the claim holds for $c=1$ since $M(i,i+1)=t_{i+1}-t_i$. Assume $h\ge 2$ and that the claim holds for $1,\ldots,h-1$. We distinguish between the cases of an odd h and an even h. In both cases assume that $j-i=h$ for some $1\le i<j\le n$.

An odd h.

$$\begin{aligned}
M(i,j) &\le M\left(i,(i+j-1)/2\right)+M\left((i+j+1)/2,j\right)+2t_j-t_{(i+j+1)/2}-t_i\\
&\le c(t_{(i+j-1)/2}-t_i)\log\left((h+1)/2\right)+c(t_j-t_{(i+j+1)/2})\log\left((h+1)/2\right)+2(t_j-t_i)\\
&\le c(t_j-t_i)\log\left((h+1)/2\right)+2(t_j-t_i)\\
&\le c(t_j-t_i)\log(h+1)-(c-2)(t_j-t_i)\\
&\le c(t_j-t_i)\log(h+1).
\end{aligned}$$

The first inequality is based on (9). The second inequality is by the induction hypothesis and by the fact that $t_{(i+j+1)/2}>t_i$. The third inequality is valid since $t_j-t_i\ge(t_{(i+j-1)/2})-(t_i+t_j-t_{(i+j+1)/2})$. The fourth inequality is implied since $\log((h+1)/2)=\log(h+1)-1$. Finally, the last inequality holds for $c\ge 2$.

An even h.

$$\begin{aligned}
M(i,j) &\le M\left(i,(i+j-2)/2\right) + M\left((i+j)/2,j\right) + 2t_j - t_{(i+j)/2} - t_i \\
&\le c(t_{(i+j-2)/2} - t_i)\log\left(h/2\right) + c(t_j - t_{(i+j)/2})\log\left((h+2)/2\right) + 2(t_j - t_i) \\
&\le c(t_j - t_i)\log\left((h+2)/2\right) + 2(t_j - t_i) \\
&\le c(t_j - t_i)\log(h+2) - (c-2)(t_j - t_i) \\
&\le c(t_j - t_i)\log(h+1) + 0.5c(t_j - t_i) - (c-2)(t - j - t_i) \\
&\le c(t_j - t_i)\log(h+1) - (0.5c-2)(t - j - t_i) \\
&\le c(t_j - t_i)\log(h+1).
\end{aligned}$$

The first inequality is based on (9). The second inequality is by the induction hypothesis and by the fact that $t_{(i+j)/2} > t_i$. The third inequality is valid since $t_j - t_i \ge (t_{(i+j-1)/2} - t_i) + (t_j - t_{(i+j+1)/2})$ and $\log((h+2)/2) \ge \log(h/2)$. The fourth inequality is implied since $\log((h+2)/2) = \log(h+2) - 1$. The fifth inequality is due to the fact that $\log_2(h+2) \le \log_2(h+1) + 0.5$ for $h \ge 2$. Rearranging terms implies the sixth inequality. Finally, the last inequality holds for $c \ge 4$. ∎

3.2. Optimal full cost. The optimal algorithm for full cost uses the optimal algorithm for merge cost as a subroutine. Let $t_1, t_2, \ldots, t_n$ be a sequence of arrivals and let L be the length of a full stream. We know that a full stream must begin at t_1; then there are two possible cases in an optimal solution. Either all the remaining streams merge to this first stream or there is a next full stream t_k for some $k \le n$. In the former case, the optimal full cost is simply $L + M(1,n)$. In the latter case, the optimal full cost is $L + M(1,k-1)$ plus the optimal full cost of the remaining arrivals $t_k, \ldots, t_n$. In both cases, the last arrival to merge to the first stream must be within $L-1$ of the first stream. That is, in the former case $t_n - t_1 \le L-1$ and in the latter case $t_{k-1} - t_1 \le L-1$.

For $1 \le i \le n$, define $G(i)$ to be the optimal full cost for the last $n-i+1$ arrivals $t_i, \ldots, t_n$. By the analysis above, we can define $G(n+1) = 0$ and for $1 \le i \le n$

$$G(i) = L + \min\left\{M(i,k-1) + G(k) : i < k \le n+1 \text{ and } t_{k-1} - t_i \le L-1\right\}. \tag{11}$$

The order of computation is $G(n+1), G(n), \ldots, G(1)$. The optimal full cost is $G(1)$. This analysis leads us to the following theorem.

THEOREM 3.3. *An optimal L-forest can be computed in time $O(nm)$ where m is the average number of arrivals in an interval of length $L-1$ that begins with an arrival.*

Proof. We begin by giving an algorithm for computing the optimal full cost and then show how it yields an algorithm to construct an optimal merge forest. By Lemma 2.6 this optimal merge forest is an L-forest. The optimal full cost algorithm proceeds in two phases. In the first phase we compute the optimal merge cost $M(i,j)$ for all i and j such that $0 \le t_j - t_i \le L-1$, so that these values can be used to compute $G(i)$. In the second phase we compute $G(i)$ from $i = n$ down to 1 using (11). Define m_i to be the cardinality of the set $\{j : 0 \le t_j - t_i \le L-1\}$ and define m to be the average of the m_i's, that is, $m = \sum_{i=1}^{n} m_i/n$. The quantity m can be thought of as the average number of arrivals in an interval of length $L-1$ that begins with an arrival.

We argue that each of the two phases can be computed in $O(nm) = O\left(\sum_{i=1}^{n} m_i\right)$ time. This is mostly a data structure issue because the number of additions and subtractions in the two phases is bounded by a constant times $\sum_{i=1}^{n} m_i$. To facilitate

the computations we define an array $A[1..n]$ of arrays. The array $A[i]$ is indexed from 0 to $m_i - 1$. Ultimately, the array entry $A[i][d]$ will contain $M(i, i+d)$ for $1 \le i \le n$ and $0 \le d < m_i$. Initially, $A[i][0] = 0$ and $A[i][d] = \infty$ for $1 \le d < m_i$. In phase one, a dynamic program based on (9) can be used to compute the ultimate value of $A[i][d] = M(i, i+d)$. Here, a specific order is required for the computation of all the nm entries in the array $A[1..n]$. Using the monotonicity property, it can be done in time $O(nm)$. In phase two, we use the array A to access the value $M(i, k-1)$ when it is needed. Since the minimization in (11) ranges over at most m_i values, the time of phase two is bounded by a constant times $\sum_{i=1}^{n} m_i$. Hence both phases together run in time $O(nm)$.

We have already seen how to construct an optimal merge tree, so all that is left is to identify the full streams. This is done inductively using the values $G(i)$ for $1 \le i \le n$ that we have already computed. We know that t_1 is a full stream. Suppose that we know the first $j \ge 1$ full streams that are indexed $f_1, f_2, \ldots, f_j$. We want to determine if f_j is the last full stream, or that the next full stream is indexed f_{j+1}. Find the smallest k such that

$$G(f_j) = L + M(f_j, k-1) + G(k),$$

where $f_j < k \le n+1$ and $t_{k-1} - t_{f_j} \le L - 1$. If $k = n+1$, then f_j is the last full stream. If $k < n+1$, then the next full stream is indexed $f_{j+1} = k$. When we are done, suppose there are s full streams which start at the arrivals indexed $f_1, f_2, \ldots, f_s$. We then compute s merge trees where the ith merge tree is for inputs $t_{f_i}, \ldots, t_{f_{i+1}-1}$ if $i < s$ and for inputs $t_{f_s}, \ldots, t_n$ if $i = s$. Given that $G(i)$ for $1 \le i \le n+1$ and $M(i,j)$ and $r(i,j)$ for $t_j - t_i \le L-1$ are already computed, then the time to compute the sequence $f_1, f_2, \ldots, f_s$ and to compute the merge trees rooted at these arrivals is $O(\sum_{i=s}^{n} m_{f_i})$ which is $O(n)$. □

We now compute an upper bound on the full cost of an arrival sequence $t_1, t_2, \ldots, t_n$, where $n \ge 2$. This time we are looking for an upper bound that depends only on $N = t_n - t_1$, n, and L and not on the sequence itself. Define $\rho = n/N$ to be the density of the n arrivals. We have $0 < \rho \le 1$. If ρ is near zero, then there are very few arrivals over the span N, so we would expect the optimal full cost to be $O(nL)$. On the other hand, if ρ is large, we would expect a lot of merging to occur, reducing the full cost considerably. This intuition is quantified in the following theorem.

THEOREM 3.4. *The optimal full cost is $O(nL)$ for any values of n and N. The optimal full cost is $O(N \log(\rho L))$ for $\rho \ge \alpha/L$ for some positive constant α.*

Proof. The first statement of the theorem is true for any solution since in the worst case each arrival gets a full stream for a total cost of nL. This is optimal if any two arrivals are more than L apart.

To prove the second statement of the theorem, assume that the optimal full cost is obtained by the L-forest F that contains s L-trees. Let the cardinalities of these trees be $m_1, m_2, \ldots, m_s$, where $m_i \ge 1$ for $1 \le i \le s$ and $\sum_{i=1}^{s} m_i = n$. It follows that

$$\text{Fcost}(F) = sL + \sum_{i=1}^{s} \text{Mcost}(m_i).$$

By Theorem 3.2 there is a constant c ($c = 4 \log_2 e$ is sufficient) such that

$$\text{Fcost}(F) \le sL + cL \sum_{i=1}^{s} \log_e m_i.$$

The convexity of the function $\log_e$ implies that

$$\text{Fcost}(F) \le sL + cL\sum_{i=1}^{s}\log_e(n/s) = sL + csL\log_e(n/s) = sL(c\log_e(n/s)+1). \tag{12}$$

The expression $sL(c\log_e(n/s)+1)$ as a function of s is concave and, by calculus, achieves a global maximum of $ce^{-1+1/c}nL$ at $s = e^{-1+1/c}n$.

We now show a natural upper bound on s:

$$s \le \frac{4N}{L}. \tag{13}$$

To see this we argue that there cannot be three full streams in an interval of length $L/2$. In such a case, the last arrival of the second tree is less than $L/2$ slots far from the root of the first tree. One could save cost by merging the root of the second tree to the root of the first tree. The worst case happens when every $L/4+1$ slots there is a new stream.

We conclude the proof of the second statement by letting $\alpha = 4e^{1-1/c}$ and assuming that $\rho \ge \alpha/L$. It follows that $e^{-1+1/c}n \ge 4N/L$. The concavity of the Fcost as a function of s implies that $\text{Fcost}(F)$ is bounded above when $s = 4N/L$ which is the maximum value for s by (13). By plugging this value for s into (12) we get

$$\begin{aligned}\text{Fcost}(F) &\le \frac{4N}{L}L\left(c\log_e\frac{n}{4N/L}+1\right)\\ &= O(N\log(\rho L)).\end{aligned}$$ □

The statement of Theorem 3.4 seems to be different from the statement of Theorem 3.2. Here the performance depends on L as well. Nevertheless, when ρ is large, the term ρL, which is close to the average number of arrivals in an interval of length L, is analogous to n.

We conclude this section by comparing analytically the performance of the traditional batching with the performance of the optimal full cost using the upper bound of Theorem 3.4.

THEOREM 3.5. *There is a positive constant α such that if $\rho \ge \alpha/L$, then batching with stream merging is $\Omega(\rho L/\log(\rho L))$ better than batching alone.*

Proof. Choose α by Theorem 3.4. The cost of batching the n full streams is nL. Therefore, by Theorem 3.4, the ratio between the performance of batching alone and batching with stream merging is

$$\Omega\left(\frac{nL}{N\log(\rho L)}\right) = \Omega\left(\frac{\rho L}{\log(\rho L)}\right).$$ □

If $\rho < \alpha/L$, then the gain is at most a constant factor in using batching with stream merging over batching alone.

4. Limited buffer size. In this section we show how to adapt our solution to the case in which each client has a limited buffer size for storing later parts of streams. Let B be the maximum buffer size. Clients start viewing the stream immediately while being able to receive data from at most two streams. Therefore, if a client has b parts in its buffer it must have viewed the first b parts of the stream. Hence, clients never need a buffer of size more than $\lfloor L/2\rfloor$. In this section, we assume that $B < \lfloor L/2\rfloor$ and we modify the algorithms accordingly.

Suppose that arrival x belongs to the merge tree T that is rooted at $r < x$. Assume further that T is an L-tree (the length of all nonroot nodes in the tree is less than or equal to L). Our goal is to calculate $b(x)$, the buffer size required by clients that arrive at time x. These clients base their receiving procedure only on earlier arrivals; therefore in calculating $b(x)$, it is enough to consider the merge tree T without all the arrivals after x. Define $T(x)$ to be this tree; in particular, x is the last arrival in $T(x)$.

LEMMA 4.1. *The buffer size required by clients arriving at x in the merge L-tree T rooted at r is*

$$b(x) = \min\{x - r, L - (x - r)\}.$$

Proof. We distinguish between the following two cases.

Case 1. Assume $0 < x - r \leq \lfloor L/2 \rfloor$. Let y be the ancestor of x in $T(x)$ (could be x itself) that is the child of the root r. It follows that x merges to r at time $y + \ell(y)$ that is the end time of the stream that was initiated at y. Now, $\ell(y) = (x-y)+(x-r)$ by (3) which implies that x merges to r at time $2x-r$. At this time x spent $(2x-r)-x = x-r$ slots receiving data from two streams and from this time on x receives data from only one stream. Hence, $b(x) = x - r$.

Case 2. Assume $\lfloor L/2 \rfloor < x - r \leq L - 1$. By the assumption T is an L-tree and hence the length of all the ancestors of x is strictly less than L. It follows that x receives the Lth part of the stream from the root and stops buffering after time $r + L$ which is the end time of the stream initiated at the root. This implies that x buffers exactly $L + r - x$ parts of the stream.

The lemma follows since if $x - r \leq \lfloor L/2 \rfloor$, then $x - r < L - (x - r)$ and if $\lfloor L/2 \rfloor < x \leq L - 1$, then $L - (x - r) < x - r$. ∎

We are now ready to describe the optimal algorithm for the full cost assuming $B < \lfloor L/2 \rfloor$. Let $t_1, t_2, \ldots, t_n$ be the sequence of arrivals. The only change is in the definition of $G(i)$ in (11). In this equation the search for the minimum value was for $i < k \leq n + 1$ such that $t_{k-1} - t_i \leq L - 1$. In what follows we modify this condition.

LEMMA 4.2. *For $1 \leq i \leq i' \leq n$, let $t_{i'}$ be the last arrival that can merge to t_i assuming t_i is a root. Then either* (i) $t_{i'} - t_i \leq B$, *or* (ii) $L - (t_{i'} - t_i) \leq B$ *and there are no arrivals t_j such that $t_i + B < t_j < t_i + L - B$.*

Proof. Assume first that there exists an arrival t_j such that $t_i + B < t_j < t_i + L - B$. By Lemma 4.1 it follows that if t_j belongs to a tree rooted at t_i then its buffer size must be greater than B. This means that t_j must belong to a tree rooted at a later arrival than t_i. Thus, in this case $t_{i'} - t_i \leq B$. Assume now that there are no arrivals t_j such that $t_i + B < t_j < t_i + L - B$. The same lemma implies that arrivals in the range $[t_i + L - B, t_i + L - 1]$ need buffer size less than or equal to B if they merge to a tree rooted at t_i. Hence, if there exists an arrival t_j such that $t_i + L - B \leq t_j \leq t_i + L - 1$, then $L - (t_{i'} - t_i) \leq B$. Otherwise, $t_{i'} - t_i \leq B$ since arrivals after $t_i + L - 1$ cannot merge to a tree rooted at t_i. ∎

Let $t_{i'}$ be the last arrival that can belong to a merge L-tree rooted at t_i as implied by Lemma 4.2. Define $G_B(i)$ to be the optimal full cost for the last $n - i + 1$ arrivals $t_i, \ldots, t_n$. We can define $G_B(n + 1) = 0$ and for $1 \leq i \leq n$

$$G_B(i) = L + \min\{M(i, k-1) + G_B(k) : i < k \leq n + 1 \text{ and } t_{k-1} \leq t_{i'}\}. \tag{14}$$

The order of computation is $G_B(n + 1), G_B(n), \ldots, G_B(1)$. The optimal full cost is $G(1)$. The following theorem is a modification of Theorem 3.3.

THEOREM 4.3. *An optimal merge forest can be computed in time $O(nm)$ where m is the average number of arrivals in an interval of length B that begins or ends with an arrival.*

Proof. The proof is almost identical to the proof of Theorem 3.3. The only change is the definition of m. The $O(nm)$ is true since by Lemma 4.2 the search for the minimum in computing $G_B(i)$ is conducted in at most in two intervals each of size B. □

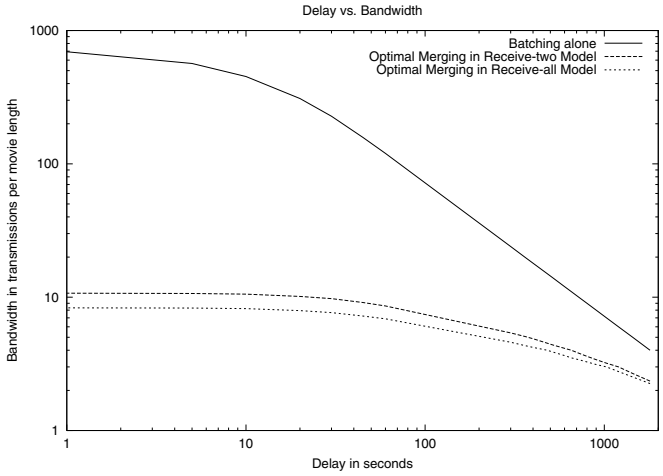


FIG. 6. *Comparison of bandwidth required for batching, batching in the receive-two model, and batching in the receive-all model. The figure plots the bandwidth requirement vs. delay for a* 2*-hour movie, with Poisson arrivals averaging every* 10 *seconds.*

5. The receive-all model. In this section we consider the receive-all model. In this model a client is capable of receiving data from all the existing streams. Surprisingly, the gain is very little compared to the receive-two model. Experimentally these two models are compared with the traditional batching (receive-one) in Figure 6. We added batching in the receive-all model to the plot of Figure 3. The figure speaks for itself. The rest of the section is devoted to demonstrating analytical comparisons of the full cost. In particular, we show a gain of at most 2. Recall that the gain from the traditional batching to the receive-two model is $\Omega(\rho L/\log(\rho L))$ (for $\rho \geq \alpha/L$ for some α).

We omit some of the details in the proofs of our claims in this section since the proofs are very similar to those in the receive-two model. We first prove some preliminary results as we did in the receive-two model.

In the receive-all model we define merge trees in exactly the same way as the receive-two model. Without going into detail, if $x_0, x_1, \ldots, x_k$ is the path from the root x_0 to node x_k that is the arrival time of a specific client, then the client x_k can receive data from all the streams $x_0, \ldots, x_k$. As in the receive-two model each stream starts at the beginning and runs continuously until it terminates, perhaps early.

Given a merge tree T and a node x, define $\ell_\omega(x)$ to be the minimum length needed to guarantee that all the clients can receive the stream using the receive-all stream merging rules.

LEMMA 5.1. *Let $x \neq r(T)$ be a nonroot node in a tree T. Then*

$$\ell_\omega(x) = z(x) - p(x). \tag{15}$$

In particular, if x is a leaf, then $\ell(x) = x - p(x)$ since $z(x) = x$.

Proof. Let a path from the root r to a leaf y be $r = x_0, x_1, \ldots, x_k = y$. At time x_k the clients arriving at y receive the following parts of the stream: part 1 from x_k, part $1 + (x_k - x_{k-1})$ from x_{k-1}, part $1 + (x_k - x_{k-2})$ from x_{k-2}, and in general part $1 + (x_k - x_i)$ from x_i. In particular they receive part $1 + (x_k - x_0)$ from the root. This means that the stream at x_k must last for at least $x_k - x_{k-1}$ slots in order for the clients arriving at y to receive parts $[1, (x_k - x_{k-1})]$. Since the only stream that can provide parts $[1 + (x_k - x_{k-1}), (x_k - x_{k-2})]$ to these clients is the one at x_{k-1}, this stream must last for at least $x_k - x_{k-2}$ slots. In general, since the stream at x_i provides parts $[1 + (x_k - x_i), (x_k - x_{i-1})]$ to these clients, this stream must last for at least $x_k - x_i$ slots.

Now let x be a node in the tree, let $p(x)$ be its parent, and let $z(x)$ be the node representing the last arrival in the subtree rooted at x. The above arguments imply that the stream at x provides parts $[1 + (z(x) - x), (z(x) - p(x))]$ to the clients arriving at $z(x)$. Since a stream is always a prefix of the full transmission, the length of the stream at x must be at least $z(x) - p(x)$. The proof is completed, since by the definition of $z(x)$, no other clients require later parts from the stream at x. ∎

Define $\text{Mcost}_\omega(T)$ to be the sum of $\ell_\omega(x)$ for all x in T except the root. For a merge forest F consisting of merge trees $T_1, \ldots, T_s$, define $\text{Fcost}_\omega(F) = s \cdot L + \sum_{i=1}^{s} \text{Mcost}_\omega(T_i)$, where L is the length of a full stream. Again we have an elegant recursive formula for the merge cost.

LEMMA 5.2. *Let T be a merge tree with root r and last stream z and let x be the last stream to merge to the root of T. Then we have*

$$\text{Mcost}_\omega(T) = \text{Mcost}_\omega(T') + \text{Mcost}_\omega(T'') + (z - r), \tag{16}$$

where T' is the subtree of all arrivals before the last stream to merge to the root of T and T'' is the subtree rooted at the last stream to merge to the root.

Proof. The length of any node in T' and T'' is the same as its length in T. Since the root of T' is the root of T, it follows that x is the only node in $\text{Mcost}(T)$ whose length is not included in $\text{Mcost}(T')$ or $\text{Mcost}(T'')$. The lemma follows, since by Lemma 5.1 the length of x is $z(x) - p(x) = z - r$. ∎

We are now ready to compute the merge cost and then the full cost. Let $t_1, \ldots, t_n$ be a sequence of arrivals. Define $M_\omega(i, j)$ to be the minimum cost of a merge tree in the receive-all model for the input sequence $t_i, \ldots, t_j$. Similar to the way we computed $M(i, j)$ we can compute $M_\omega(i, j)$ using the recursive formulation based on (16):

$$M_\omega(i, j) = \min_{i<k\leq j} \{M_\omega(i, k-1) + M_\omega(k, j)\} + (t_j - t_i) \tag{17}$$

with the initialization $M_\omega(i, i) = 0$ for $1 \leq i \leq n$. The optimal cost for the entire sequence is $M_\omega(1, n)$. Because t_k does not appear as a parameter in (17), we can use the simpler approach of Yao [45] to show monotonicity. As a result we have an $O(n^2)$ time algorithm for computing an optimal merge tree in the receive-all model.

Equations (9) and (17) allow us to give bounds on the gain in optimal merge cost that can be achieved by moving to the receive-all model.

THEOREM 5.3. *For any arrival sequence $t_1, \ldots, t_n$ and $1 \leq i \leq j \leq n$*

$$M_\omega(i, j) \leq M(i, j) \leq 2M_\omega(i, j).$$

Proof. The proof is by induction on $j-i$. If $j-i=0$, then $M(i,j)=M_\omega(i,j)=0$. If $j-i>0$, then let k and h be such that

$$M(i,j) = M(i,k-1) + M(k,j) + 2t_j - t_k - t_i, \tag{18}$$

$$M_\omega(i,j) = M_\omega(i,h-1) + M_\omega(h,j) + t_j - t_i. \tag{19}$$

The following proves the lower bound claim of the theorem.

$$\begin{aligned} M_\omega(i,j) &\le M_\omega(i,k-1) + M_\omega(k,j) + t_j - t_i \\ &\le M(i,k-1) + M(k,j) + t_j - t_i \\ &\le M(i,k-1) + M(k,j) + 2t_j - t_k - t_i \\ &\le M(i,j). \end{aligned}$$

The first inequality follows since $M_\omega(i,j)$ is a minimization. By the induction hypothesis, we get the second inequality. The third inequality is implied since $t_j \ge t_k$. Finally, (18) yields the last inequality.

The following proves the upper bound claim of the theorem:

$$\begin{aligned} M(i,j) &\le M(i,h-1) + M(h,j) + 2t_j - t_h - t_i \\ &\le 2M_\omega(i,h-1) + 2M_\omega(h,j) + 2t_j - t_h - t_i \\ &\le 2M_\omega(i,h-1) + 2M_\omega(h,j) + 2(t_j - t_i) \\ &\le 2M_\omega(i,j). \end{aligned}$$

The first inequality follows since $M(i,j)$ is a minimization. By the induction hypothesis, we get the second inequality. The third inequality is implied since $t_h \ge t_i$. Finally, (19) yields the last inequality. □ In the same way as we did for the receive-two model, define $G_\omega(i)$ to be the optimal full cost in the receive-all model for the sequence $t_i, \ldots t_n$, the last $n-i+1$ arrivals. We have $G(n+1)=0$ and for $1 \le i \le n$

$$G_\omega(i) = L + \min\{M_\omega(i,k-1) + G_\omega(k) : i < k \le n+1 \text{ and } t_{k-1} - t_i \le L-1\}. \tag{20}$$

The order of computation is $G_\omega(n+1), G_\omega(n), \ldots, G_\omega(1)$. The optimal full cost is $G_\omega(1)$. Using exactly the same technique as we did for the receive-two model, we achieve an $O(nm)$ algorithm for computing an optimal merge forest in the receive-all model, where m is the average number of arrivals in a interval of length $L-1$ that begins with an arrival.

We now apply Theorem 5.3 to show the same factor of 2 bound on the optimal full cost.

THEOREM 5.4. *For any arrival sequence $t_1, \ldots, t_n$ and $1 \le i \le n+1$,*

$$G_\omega(i) \le G(i) \le 2G_\omega(i).$$

Proof. The proof is by a reverse induction from $n+1$ to 1. For $n+1$ we have $G_\omega(n+1) = G(n+1) = 0$. If $i < n+1$, then we proceed in a way similar to the proof of theorem 5.3, by letting $k > i$ and $h > i$ be such that $t_{k-1} - t_i \le L-1$, $t_{h-1} - t_i \le L-1$, and

$$G(i) = L + M(i,k-1) + G(k), \tag{21}$$

$$G_\omega(i) = L + M_\omega(i,h-1) + G_\omega(h). \tag{22}$$

The following proves the lower bound claim of the theorem:

$$\begin{aligned} G_\omega(i) &\le L + M_\omega(i, k-1) + G_\omega(k) \\ &\le L + M(i, k-1) + G(k) \\ &\le G(i). \end{aligned}$$

The first inequality follows since $G_\omega(i)$ is a minimization. The induction hypothesis and Theorem 5.3 imply the second inequality. The last inequality is by (21).

The following proves the upper bound claim of the theorem:

$$\begin{aligned} G(i) &\le L + M(i, h-1) + G(h) \\ &\le 2L + 2M_\omega(i, h-1) + 2G_\omega(h) \\ &\le 2G_\omega(i). \end{aligned}$$

The first inequality follows since $G(i)$ is a minimization. The induction hypothesis, Theorem 5.3, and the fact that L is positive imply the second inequality. The last inequality is by (22). □

The factor of 2 is not at all tight for optimal full cost. For example, if $L = 2$, then it can be shown that the optimal full cost in the receive-two model is identical to the optimal full cost in the receive-all model.

6. Conclusions. In this paper, we addressed the problem of designing efficient off-line algorithms to compute the optimal stream merging in media-on-demand systems. In a stream merging system, clients are assigned to receive data from streams that transmit a popular media where they are capable of receiving data from two streams simultaneously. When clients arrive, they get a receiving program that instructs them from which streams to receive data and when. Their program is independent of later arrivals and is simple. Streams are broadcast by the server each time clients request to view the transmission. However, very few of the streams are full streams, thus allowing the savings in the total utilized bandwidth per one popular media. The main advantage of stream merging is its flexibility. With stream merging, it is easier to allocate channels dynamically to various media based on the current demand.

The main objective of this paper was to construct an efficient optimal algorithm. Recall that n is the number of arrivals, L is the length of the full stream, and m is the average number of arrivals in an interval of length $L - 1$ that starts with an arrival. With these parameters, we have an $O(nm)$ optimal algorithm. We also showed how to modify our algorithm to be optimal even if the buffer of clients is limited in its size while maintaining the same running time complexity. To obtain these results, we introduced a new abstract model for the stream merging paradigm. Our merge forest model captures all the information regarding the system. We first analyze a single merge tree and then the merge forest itself.

Finally, we considered a stronger model in which clients may receive data from all the existing streams simultaneously. We showed how our techniques extend to this model with less effort. We used these results to show that analytically the gain from the receive-two model to the receive-all model is at most 2, whereas the gain from the traditional batching to the receive-two model is of order $\rho L / \log(\rho L)$, where $\rho \le 1$ is the density of the n arrivals in the time interval between the first and the last arrival and ρ is large enough. This phenomenon was known experimentally and we support it with analytical results.

REFERENCES

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, *Design and analysis of permutation-based pyramid broadcasting*, Multimedia Systems, 7 (1999), pp. 439–448.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, *Adaptive piggybacking schemes for video-on-demand systems*, Multimedia Tools Appl., 16 (2002), pp. 231–250.

[3] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, *The maximum factor queue length batching scheme for video-on-demand systems*, IEEE Trans. Computers, 50 (2001), pp. 97–110.

[4] A. Bar-Noy, J. Goshi, R. E. Ladner, and K. Tam, *Comparison of stream merging algorithms for media-on-demand*, Multimedia Systems, 9 (2004), pp. 411–423.

[5] A. Bar-Noy and R. E. Ladner, *Competitive on-line stream merging algorithms for media-on-demand*, J. Algorithms, 48 (2003), pp. 59–90.

[6] A. Borchers and P. Gupta, *Extending the quadrangle inequality to speed-up dynamic programming*, Inform. Process. Lett., 49 (1994), pp. 287–290.

[7] Y. Cai and K. A. Hua, *An efficient bandwidth-sharing technique for true video on demand systems*, in Proceedings of the 7th ACM International Conference on Multimedia, 1999, pp. 211–214.

[8] Y. Cai, K. A. Hua, and K. Vu, *Optimizing patching performance*, in Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN '99), 1999, pp. 204–215.

[9] S. W. Carter and D. D. E. Long, *Improving video-on-demand server efficiency through stream tapping*, in Proceedings of the 6th International Conference on Computer Communications and Networks (ICCCN '97), 1997, pp. 200–207.

[10] S. W. Carter and D. D. E. Long, *Improving bandwidth efficiency of video-on-demand servers*, Computer Networks, 31 (1999), pp. 111–123.

[11] W. Chan, T. Lam, H. Ting, and W. Wong, *On-line stream merging in a general setting*, Theoret. Comput. Sci., 296 (2003), pp. 27–46.

[12] W. Chan, T. Lam, H. Ting, and W. Wong, *Competitive analysis of on-line stream merging algorithms*, in Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS), 2002, pp. 188–200.

[13] T. Chiueh and C. Lu, *A periodic broadcasting approach to video-on-demand service*, in Proceedings of the SPIE Conference on Multimedia Computing and Networking (MMCN '95), 1995, pp. 162–169.

[14] E. G. Coffman, Jr., P. Jelenković, and P. Momčilović, *The dyadic stream merging algorithm*, J. Algorithms, 43 (2002), pp. 120–137.

[15] A. Dan, D. Sitaram, and P. Shahabuddin, *Dynamic batching policies for an on-demand video server*, Multimedia Systems, 4 (1996), pp. 112–121.

[16] D. L. Eager, M. Ferris, and M. K. Vernon, *Optimized regional caching for on-demand data delivery*, in Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN '99), 1999, pp. 301–316.

[17] D. L. Eager and M. K. Vernon, *Dynamic skyscraper broadcasts for video-on-demand*, in Proceedings of the 4th International Workshop on Advances in Multimedia Information Systems (MIS '98), 1998, pp. 18–32.

[18] D. L. Eager, M. K. Vernon, and J. Zahorjan, *Minimizing bandwidth requirements for on-demand data delivery*, IEEE Trans. Knowl. Data Engrg., 13 (2001), pp. 742–757.

[19] D. L. Eager, M. K. Vernon, and J. Zahorjan, *Optimal and efficient merging schedules for video-on-demand servers*, in Proceedings of the 7th ACM International Multimedia Conference, 1999, pp. 199–203.

[20] L. Gao, J. Kurose, and D. Towsley, *Efficient schemes for broadcasting popular videos*, Multimedia Systems, 8 (2002), pp. 284–294.

[21] L. Gao and D. Towsley, *Supplying instantaneous video-on-demand services using controlled multicast*, in Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS '99), 1999.

[22] L. Gao, Z. Zhang, and D. Towsley, *Catching and selective catching: Efficient latency reduction techniques for delivering continuous multimedia streams*, in Proceedings of the 7th ACM International Conference on Multimedia, 1999, pp. 203–206.

[23] L. Golubchik, J. C. S. Liu, and R. R. Muntz, *Reducing I/O demand in video-on-demand storage servers*, in Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '95), 1995, pp. 25–36.

[24] L. Golubchik, J. C. S. Liu, and R. R. Muntz, *Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers*, Multimedia Systems, 4 (1996), pp. 140–155.

[25] K. A. Hua, Y. Cai, and S. Sheu, *Patching: A multicast technique for true video-on-demand services*, in Proceedings of the 6th ACM International Conference on Multimedia, 1998, pp. 191–200.

[26] K. A. Hua, Y. Cai, and S. Sheu, *Exploiting client bandwidth for more efficient video broadcast*, in Proceedings of the 7th International Conference on Computer Communications and Networks (ICCCN '98), 1998, pp. 848–856.

[27] K. A. Hua and S. Sheu, *Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems*, in Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 1997, pp. 89–100.

[28] L. Juhn and L. Tseng, *Harmonic broadcasting for video-on-demand service*, IEEE Trans. Broadcasting, 43 (1997), pp. 268–271.

[29] L. Juhn and L. Tseng, *Staircase data broadcasting and receiving scheme for hot video service*, IEEE Trans. Consumer Electronics, 43 (1997), pp. 1110–1117.

[30] L. Juhn and L. Tseng, *Fast broadcasting for hot video access*, in Proceedings of the 4th International Workshop on Real-Time Computing Systems and Applications (RTCSA '97), 1997, pp. 237–243.

[31] L. Juhn and L. Tseng, *Enhancing harmonic data broadcasting and receiving scheme for popular video service*, IEEE Trans. Consumer Electronics, 44 (1998), pp. 343–346.

[32] L. Juhn and L. Tseng, *Fast data broadcasting and receiving scheme for popular video service*, IEEE Trans. Broadcasting, 44 (1998), pp. 100–105.

[33] D. E. Knuth, *Optimum binary search trees*, Acta Informa., 1 (1971), pp. 14–25.

[34] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Searching and Sorting*, 2nd ed., Addison–Wesley, Reading, MA, 1998.

[35] S. W. Lau, J. C. S. Liu, and L. Golubchik, *Merging video streams in a multimedia storage server: Complexity and heuristics*, Multimedia Systems, 6 (1998), pp. 29–42.

[36] J. Pâris, S. W. Carter, and D. D. E. Long, *A low bandwidth broadcasting protocol for video on demand*, in Proceedings of the 7th International Conference on Computer Communications and Networks (ICCCN '98), 1998, pp. 690–697.

[37] J. Pâris, S. W. Carter, and D. D. E. Long, *A hybrid broadcasting protocol for video on demand*, in Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN '99), 1999, pp. 317–326.

[38] J. Pâris and D. D. E. Long, *Limiting the receiving bandwidth of broadcasting protocols for video-on-demand*, in Proceedings of the Euromedia Conference, 2000, pp. 107–111.

[39] J. Pâris, D. D. E. Long, and P. E. Mantey, *Zero-delay broadcasting protocols for video on demand*, in Proceedings of the 7th ACM International Conference on Multimedia, 1999, pp. 189–197.

[40] S. Sen, L. Gao, J. Rexford, and D. Towsley, *Optimal patching schemes for efficient multimedia streaming*, in Proceedings of the 9th IEEE International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '99), 1999.

[41] S. Sen, L. Gao, and D. Towsley, *Frame-based Periodic Broadcast and Fundamental Resource Tradeoffs*, Technical report 99–78, University of Massachusetts, Amherst, MA.

[42] Y. Tseng, C. Hsieh, M. Yang, W. Liao, and J. Sheu, *Data broadcasting and seamless channel transition for highly-demanded videos*, in Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00), 2000, pp. 4E–2.

[43] S. Viswanathan and T. Imielinski, *Pyramid broadcasting for video-on-demand service*, in Proceedings of the SPIE Conference on Multimedia Computing and Networking (MMCN '95), 1995, pp. 66–77.

[44] S. Viswanathan and T. Imielinski, *Metropolitan area video-on-demand service using pyramid broadcasting*, Multimedia Systems, 4 (1996), pp. 197–208.

[45] F. F. Yao, *Efficient dynamic programming using quadrangle inequalities*, in Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC '80), 1980, pp. 429–435.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1035–1051

BETTER ONLINE ALGORITHMS FOR SCHEDULING WITH MACHINE COST*

GYÖRGY DÓSA† AND YONG HE‡

Abstract. For most scheduling problems the set of machines is fixed initially and remains unchanged for the duration of the problem. Recently Imreh and Noga proposed adding the concept of machine cost to scheduling problems and considered the so-called *list model* problem. For this problem, we are given a sequence of independent jobs with positive sizes, which must be processed nonpreemptively on a machine. No machines are initially provided, and when a job is revealed the algorithm has the option to purchase new machines. The objective is to minimize the sum of the makespan and cost of machines. In this paper, we first present an online algorithm with a competitive ratio at most 1.5798, which improves the known upper bound 1.618. Then for a special case where every job size is no greater than the machine cost, we present an optimal online algorithm with a competitive ratio 4/3. Last, we present an algorithm with a competitive ratio at most 3/2 for the semionline problem with known largest size, which improves the known upper bound 1.5309.

Key words. parallel machine scheduling, machine cost, online algorithm, competitive analysis

AMS subject classifications. 90B35, 90C27

DOI. 10.1137/S009753970343395X

1. Introduction.

In the classical parallel identical machine scheduling problem $P|online|C_{\max}$, we are given a sequence $\mathcal{J}$ of independent jobs with positive processing times (sizes) $p_1, p_2, \ldots, p_n$, which must be nonpreemptively scheduled on m parallel and identical machines with the objective to minimize the *makespan* $C_{\max}$ (i.e., the maximum machine load, where machine load means the total size of jobs assigned to this machine). Jobs come one by one (online over list), and it requires one to schedule jobs irrevocably to a machine as soon as they are given, without any knowledge about jobs that follow later on. There has been a great deal of work on this problem [1, 2, 3, 5, 6, 7, 8, 12, 14], but the optimal online algorithm is still unknown for $m > 3$. In [11], Imreh and Noga proposed a variant of the above problem. The differences are that (1) no machines are initially provided, (2) when a job is revealed the algorithm has the option to purchase new machines, and (3) the objective is to minimize the sum of the makespan and cost of machines. We refer to this problem as the *list model*.

This problem is quite different from the classical scheduling problems, where we typically have a fixed number m of machines, the algorithm (scheduler) makes no decision regarding the machine number nor is it allowed to change the machine number later, and the provided machines can be utilized without cost. Imreh and Noga [11] proposed adding the concept of machine cost to the scheduling problem due to the following main motivation: First, real machines have cost. Second, the performance of an algorithm on a given input can be highly dependent on the number of machines. This seems particularly true when considering worst-case measures (such as competitive analysis). Last, by considering such a variant we may find other

*Received by the editors September 2, 2003; accepted for publication (in revised form) February 24, 2004; published electronically June 25, 2004. This research was supported by the Teaching and Research Award Program for Outstanding Young Teachers in Higher Education Institutions of the MOE, China, and the National Natural Science Foundation of China (10271110, 60021201).
http://www.siam.org/journals/sicomp/33-5/43395.html

†Department of Mathematics, University of Veszprém, Hungary (dosagy@almos.vein.hu).
‡Department of Mathematics, State Key Lab of CAD & CG, Zhejiang University, Hangzhou 310027, People's Republic of China (mathhey@zju.edu.cn).

interesting problems and/or gain insight into the original.

Following [11], He and Cai [9, 4] considered a variant of the list model problem. They studied the problem in a *semionline* environment. Here semionline means that we have partial knowledge of jobs before constructing a schedule, and we still cannot rearrange any job which has been assigned to machines. If the problem is *semionline with known largest size*, then we know the size of the largest job as a priori. If the problem is *semionline with known total size*, then we know the sum of sizes of all jobs in advance. If the problem is *semionline with decreasing sizes*, then we know that jobs come in an order of nonincreasing sizes. Note that the semionline problem with known largest size can be viewed as a relaxation of the problem with decreasing sizes. These semionline versions were first proposed by different authors when the problem $P||C_{\max}$ was studied [10, 13, 8, 16].

The quality of an online or semionline algorithm A is measured by its *competitive ratio*. For any sequence I of jobs, let $A(I)$ denote the corresponding objective value of a schedule produced by A, and let $OPT(I)$ denote the optimal objective value. Then the competitive ratio of A is defined as the smallest number c such that $A(I) \leq cOPT(I)$ for all sequences. An algorithm with a competitive ratio c is called a *c-competitive algorithm*. An online (semionline) scheduling problem has a *lower bound* ρ if no online (semionline) algorithm has a competitive ratio smaller than ρ. An online (semionline) algorithm is called *optimal* if its competitive ratio matches the lower bound of the problem.

For the list model problem, Imreh and Noga [11] presented an online $(1+\sqrt{5})/2 \approx 1.618$-competitive algorithm A_ρ while the lower bound was 4/3. If randomization is allowed, Seiden [15] provided a lower bound 1.06532. For the semionline problem with known largest size, He and Cai [9] presented an algorithm with a competitive ratio at most 1.5309 while the lower bound was 4/3. For the semionline problem with known total size, He and Cai [9] presented an algorithm with a competitive ratio at most 1.414 while the lower bound was 1.161. For the semionline problem with decreasing sizes, Cai and He [4] presented an algorithm with a competitive ratio 3/2 while the lower bound was still 4/3. These semionline algorithms are essentially modified from A_ρ.

In this paper, we continue to consider the list model problem and its semionline problem with known largest size. We first present a new online algorithm with a competitive ratio at most $(2\sqrt{6}+3)/5 \approx 1.5798$, which improves the known upper bound 1.618. Then for a special case where every job has a size no greater than the machine cost (called *small job*), we present an optimal online algorithm with a competitive ratio 4/3. Last, we present a new two-phase algorithm with a competitive ratio at most 3/2 for the semionline problem with known largest size, which also improves the known result. Compared with A_ρ, our algorithms apply new strategies to decide when to purchase machines. In fact, the decision regarding when to purchase new machines depends on the current makespan, and the current objective value with respect to the lower bounds of the current optimal value, while A_ρ simply depends on the total size of all arrived jobs.

The paper is organized as follows. Section 2 presents several preliminary results. Section 3 considers the general case and small job case of the list model problem. Section 4 deals with the semionline problem with known largest size. Finally section 5 contains some remarks.

2. Preliminaries. Throughout the remainder of the paper, we will use the following notation. Denote by $L = \max_{1,2,\ldots,n}\{p_i\}$ the largest size of all jobs, by $P = \sum_{i=1}^{n} p_i$ the total size of all jobs, and by $P_k = \sum_{j=1}^{k} p_j$ the total size of the first

k jobs. By normalizing, we assume that the cost of purchasing a machine is 1. If m' is the number of machines used by an online or semionline algorithm A after all jobs are assigned, and $C_{\max}$ is its makespan, then the objective value produced by A is $A(I) = C_{\max} + m'$.

LEMMA 2.1 (see [11]). *The optimal objective value is at least* $2\sqrt{P}$. *Further, if* $L \geq \sqrt{P}$, *then the optimal objective value is at least* $L + P/L$; *or, equivalently, if there is a job* p_k *such that* $p_k \geq \sqrt{P}$, *then the optimal objective value is at least* $p_k + P/p_k$.

The following online algorithm A_ρ was presented by Imreh and Noga [11] to solve the list model problem.

ALGORITHM A_ρ. Set $\rho = \{\rho_1, \rho_2, \rho_3, \dots, \rho_i, \dots\} = \{0, 4, 9, 16, \dots, i^2, \dots\}$. When job p_i is revealed A_ρ purchases machines (if necessary) so that the current number of machines j satisfies $\rho_j \leq P_i < \rho_{j+1}$. A_ρ then assigns job p_i to the machine with the minimum current load.

In [11], Imreh and Noga have shown that A_ρ is $(1+\sqrt{5})/2$-competitive. Lemma 2.2 shows that the ratio can be tightened for a special case.

LEMMA 2.2 (see [9]). *If* $L \leq 3$, *then* A_ρ *has a competitive ratio* $3/2$.

LEMMA 2.3 (see [11, 9]). *Any online algorithm for the problem with small jobs and any semionline algorithm for the problem with known largest size have a competitive ratio at least* $4/3$.

Proof. We show the result by adversary method. Let the largest size be $L = 1/N$, where N is a sufficiently large positive integer. Consider a long sequence of jobs, each with size L. If an algorithm A never purchases a second machine, then $8N$ jobs come. It follows that $\frac{A(I)}{OPT(I)} \geq \frac{8N/N+1}{8N/(2N)+2} = \frac{3}{2}$. So we assume that A purchases a second machine when job p_i arrives. Then no new job comes. If $P_i \leq 2$, then in the optimal schedule all jobs are processed on one machine and $\frac{A(I)}{OPT(I)} \geq \frac{P_i - 1/N+2}{P_i+1} \geq \frac{4-1/N}{3}$. If $P_i > 2$, then in the optimal schedule all jobs can be split nearly evenly between two machines and $\frac{A(I)}{OPT(I)} \geq \frac{P_i - 1/N+2}{P_i/2+2} \geq \frac{4-1/N}{3+1/N}$. Since we choose N to be arbitrarily large, we obtain the result. □

Before going to the main content, we further define some notation. When analyzing an online or semionline algorithm in the later sections, we use the following:

m = the current number of machines purchased;

$m_0 = \lceil L \rceil$;

p_{i_j} = the first job which is processed on the jth machine, $j = 1, \dots, m$.

Right after the algorithm scheduling job p_k, $k = 1, 2, \dots, n$, we define

$s_{j,k}$ = the current load of the jth machine, $j = 1, 2, \dots, m$;

s_k = the minimum current machine load, i.e., $s_k = \min\{s_{j,k} | j = 1, \dots, m\}$;

C_k = the current makespan;

z_k = the current objective value, i.e., $z_k = C_k + m$;

$C_{1,k} = 2\sqrt{P_k}$, a lower bound of the current optimal objective value;

$C_{2,k} = L + \frac{P_k}{L}$, another lower bound of the current optimal objective value, where $L \geq \sqrt{P_k}$.

3. Online algorithms.

3.1. Better online algorithm for general case. This subsection considers the original list model problem. We present an algorithm $H1$ with a competitive ratio at most $\frac{2\sqrt{6}+3}{5} \approx 1.5798$. Compared with A_ρ, $H1$ uses a different strategy to decide when a new machine is purchased. It always assigns an incoming job to the machine with minimum current load as long as its new load would be no greater than $2m$,

where m is the current machine number. Otherwise, a new machine is purchased to process the incoming job. Formally, it can be described as follows.

ALGORITHM $H1$.

1. Assign p_1 to the first machine. Let $k = 2$, $m = 1$.
2. Assign p_k to the machine with minimum current load if the new load will be no more than $2m$, and go to 4. Else go to 3.
3. Assign p_k to a new machine, $m = m + 1$, go to 4.
4. Let $k = k + 1$, if $k > n$, stop. Otherwise, return 2.

LEMMA 3.1. *Let $m \geq 2$ be the number of machines currently purchased by $H1$. If the current minimum machine load satisfies $s_k \leq m - 1$, then the sum of the current machine loads is*

$$\sum_{j=1}^{m} s_{j,k} \geq s_k^2 - (m-1)s_k + (m-1)m.$$

Proof. Obviously for any $1 \leq j < l \leq m$, we have that

$$s_{j,k} + s_{l,k} \geq s_k + p_{i_l} > 2\,(l-1)$$

by the algorithm rule. Consider the following parametric nonlinear program with parameter $s_k \in [0, m-1]$:

$$\begin{array}{ll} \min & u(s_k) = \sum_{j=1}^{m} s_{j,k} \\ \text{s.t.} & s_{j,k} + s_{l,k} \geq 2\,(l-1)\,, \quad 1 \leq j < l \leq m, \\ & \min\{s_{j,k} : 1 \leq j \leq m\} = s_k. \end{array}$$

Assume that $i \leq s_k < i + 1 \leq m - 1$ for some integer $i \geq 0$. Then it is easy to see that its optimal solution is

$$s_{j,k} = \begin{cases} s_k & \text{if} \quad 1 \leq j \leq i+1, \\ 2(j-1) - s_k & \text{if} \quad i+1 < j \leq m, \end{cases}$$

and its minimum value is

$$u^*(s_k) = \begin{cases} s_k^2 - (m-1)s_k + (m-1)m & \text{if} \quad s_k = i, \\ (m+i)(m-i-1) - ms_k + 2(i+1)s_k & \text{if} \quad i < s_k < i+1. \end{cases}$$

We define the functions $f_1(s_k) = s_k^2 - (m-1)s_k + (m-1)m$ on the interval $[0, m-1]$, and $f_2(s_k) = (m+i)(m-i-1) - ms_k + 2(i+1)s_k$ on $[i, i+1]$ for each $i = 0, 1, \ldots, m-2$. Since the function $f_2(s_k)$ is linear on the interval $[i, i+1]$, and $f_2(s_k) = f_1(s_k)$ at $s_k = 0, 1, \ldots, m-1$, hence the function $u^*(s_k)$ is continuous and piecewise linear on $[0, m-1]$. Further, the function $f_1(s_k)$ is convex on $[0, m-1]$, and we conclude that $u^*(s_k) \geq f_1(s_k)$ for any $0 \leq s_k \leq m-1$. Therefore we know that $u(s_k) \geq u^*(s_k) \geq f_1(s_k) = s_k^2 - (m-1)s_k + (m-1)m$ for any $0 \leq s_k \leq m-1$. ∎

COROLLARY 3.2. *Under the same conditions as those of Lemma 3.1, we have $\sum_{j=1}^{m} s_{j,k} \geq \frac{3}{4}(m-1)^2 + (m-1) > \frac{3}{4}m(m-1)$.*

Proof. The function $s_k^2 - (m-1)s_k + (m-1)m$ attains its minimum value when $s_k = \frac{m-1}{2}$. ∎

Remark 3.1. Let $m \geq 2$ be the number of machines currently purchased by $H1$. If the minimum current machine load satisfies $s_k \geq m - 1$, then trivially $\sum_{j=1}^{m} s_{j,k} \geq (m-1)m$.

To prove the competitive ratio of $H1$, we introduce more notation. For any $l \geq 1$, denote $u_l = \max\{p_i | 1 \leq i \leq l\}$, and define

$$C'_l = \begin{cases} C_{1,l} & \text{if} \quad u_l \leq \sqrt{P_l}, \\ C_{2,l} & \text{if} \quad u_l > \sqrt{P_l}. \end{cases}$$

It is clear that $C'_l \geq C_{1,l}$ for any $l \geq 1$.

LEMMA 3.3. *C'_l is increasing with regard to l, i.e., $C'_1 \leq C'_2 \leq \cdots \leq C'_n$.*

Proof. If $C'_{l-1} = C_{1,l-1}$, then trivially $C'_l \geq C_{1,l} > C_{1,l-1} = C'_{l-1}$. Hence suppose that $C'_{l-1} = C_{2,l-1}$. If further $C'_l = C_{2,l}$, then from $u_{l-1} \geq \sqrt{P_{l-1}}$, we have $C'_l = C_{2,l} = \frac{P_l}{u_l} + u_l > \frac{P_{l-1}}{u_l} + u_l \geq \frac{P_{l-1}}{u_{l-1}} + u_{l-1} = C_{2,l-1} = C'_{l-1}$. Otherwise $C'_l = C_{1,l}$, and we know that $u_{l-1}^2 > P_{l-1}$ and $u_l^2 \leq P_l$. It follows that $C'_l = C_{1,l} = 2\sqrt{P_l} \geq 2u_l \geq u_{l-1} + u_{l-1} > u_{l-1} + \frac{P_{l-1}}{u_{l-1}} = C_{2,l-1} = C'_{l-1}$. We are done. ∎

THEOREM 3.4. *The competitive ratio of $H1$ is at most $\alpha \doteq \frac{2\sqrt{6}+3}{5} \approx 1.5798$.*

Proof. We show by induction that for every $l = 1, 2, \ldots, n$, $z_l \leq \alpha C'_l$ holds. It follows that the competitive ratio of $H1$ is at most α.

It is clear that the result is true for $l = 1$. Assuming that the result is true for $l = k - 1$, two cases are considered according to the assignment of p_k.

Case 1. p_k is assigned to one of m machines by step 2. If the makespan is unchanged (i.e., $C_k = C_{k-1}$), then $z_k = z_{k-1}$ and thus the result trivially holds for $l = k$ by the induction assumption and Lemma 3.3. Hence we assume that the makespan is increased and equal to $C_k = s_{k-1} + p_k$, and

$$z_k = C_k + m = s_{k-1} + p_k + m. \tag{3.1}$$

We distinguish three subcases according to the value of m.

Subcase 1.1. $m = 1$ or $m = 2$. From $\alpha C'_k \geq \alpha C_{1,k} \geq 3\sqrt{P_k}$ and (3.1), it suffices to show

$$s_{k-1} + p_k + m \leq 3\sqrt{P_k}. \tag{3.2}$$

If $m = 1$, then by the algorithm rule we have $s_{k-1} + p_k \leq 2m = 2$. If $s_{k-1} + p_k < 1$, then $z_k < 2$, from which it follows that in the optimal schedule there is also only one machine and thus the schedule yielded by $H1$ is optimal. If $1 \leq s_{k-1} + p_k \leq 2$, then trivially $s_{k-1} + p_k + 1 \leq 3\sqrt{s_{k-1} + p_k}$ holds, and we thus get (3.2).

If $m = 2$, then by the algorithm rule we have $s_{k-1} + p_k \leq 4$. On the other hand, right after p_{i_2} is assigned to the second machine, the sum of all job sizes is more than 2; thus the makespan $s_{k-1} + p_k$ must be greater than 1. $1 < s_{k-1} + p_k \leq 4$ implies $s_{k-1} + p_k + 2 \leq 3\sqrt{s_{k-1} + p_k}$, and thus $s_{k-1} + p_k + 2 \leq 3\sqrt{s_{k-1} + p_k} \leq 3\sqrt{P_k}$.

Subcase 1.2. $2 < m \leq s_{k-1} + 1$. If further $m \leq s_{k-1}$, then trivially $P_k \geq s_{k-1}m + p_k \geq m^2$, and thus $z_k = s_{k-1} + p_k + m \leq 2m + m \leq 3\sqrt{P_k} = \frac{3}{2}C_{1,k}$. The desired competitive ratio is obtained; hence we assume that $m - 1 \leq s_{k-1} < m$. If $p_k > s_{k-1} + 1$, then $P_k \geq s_{k-1}m + p_k \geq (m-1)\,m + m = m^2$, and again $z_k \leq \frac{3}{2}C_{1,k}$. If $p_k \leq s_{k-1}$, then by (3.1) and $\frac{1}{4} \leq \frac{s_{k-1}}{m} \leq 1$, we have that $z_k \leq 2s_{k-1} + m \leq 3\sqrt{s_{k-1}m} \leq 3\sqrt{P_k} = \frac{3}{2}C_{1,k}$. Last, if $s_{k-1} < p_k \leq s_{k-1} + 1$, then similarly $z_k \leq 2s_{k-1} + 1 + m \leq 3\sqrt{s_{k-1}m + s_{k-1}} \leq 3\sqrt{P_k} = \frac{3}{2}C_{1,k}$ since $\frac{1}{4} \leq \frac{s_{k-1}}{m+1} \leq 1$.

Subcase 1.3. $m > s_{k-1} + 1$. If $p_k \leq m - s_{k-1}$, then $z_k \leq 2m$ by (3.1). By Corollary 3.2, we have $P_k \geq \frac{3}{4}m\,(m-1)$. $m > 2$ implies $2m \leq 3\sqrt{\frac{3}{4}m\,(m-1)}$, and we thus get $z_k \leq \frac{3}{2}C_{1,k}$. Hence we assume that

$$p_k > m - s_{k-1}. \tag{3.3}$$

Combining (3.3) with Lemma 3.1, we have

$$P_k = \sum_{j=1}^{m} s_{j,k} + p_k \geq s_{k-1}^2 - (m-1)\, s_{k-1} + (m-1)\, m + p_k \tag{3.4}$$
$$> s_{k-1}^2 - m s_{k-1} + m^2.$$

If $p_k \leq \sqrt{P_k}$, then by (3.1) and (3.4), we have

$$z_k \leq \sqrt{P_k} + s_{k-1} + m \leq \sqrt{P_k} + 2\sqrt{s_{k-1}^2 - m s_{k-1} + m^2} < 3\sqrt{P_k} = \frac{3}{2} C_{1,k}.$$

Hence we assume that $p_k \geq \sqrt{P_k}$. It follows from Lemma 2.1 that $p_k + \frac{P_k}{p_k}$ $(\leq C'_k)$ is a valid lower bound of the current optimal objective value. In order to get the desired competitive ratio, from (3.1) and (3.4), it suffices to prove

$$s_{k-1} + p_k + m - \alpha \left(p_k + \frac{s_{k-1}^2 - m s_{k-1} + m^2}{p_k} \right) \leq 0. \tag{3.5}$$

To see it, we first consider the case of $\frac{s_{k-1}}{m} > \frac{3}{5}$. Since $\alpha \leq \frac{8}{5}$, $s_{k-1} + p_k \leq 2m$, and ${p_k}^2 \geq P_k > s_{k-1}^2 - m s_{k-1} + m^2$, the derivative of the left side of (3.5) with respect to p_k is

$$1 - \alpha \left(1 - \frac{s_{k-1}^2 - m s_{k-1} + m^2}{{p_k}^2} \right) > 1 - \frac{8}{5} \left(1 - \frac{s_{k-1}^2 - m s_{k-1} + m^2}{(2m - s_{k-1})^2} \right) > 0,$$

where the last inequality is equivalent to $5(\frac{s_{k-1}}{m})^2 + 4\frac{s_{k-1}}{m} - 4 > 0$, which is trivially true due to $\frac{s_{k-1}}{m} > \frac{3}{5}$. Hence we only need to show (3.5) is valid when $p_k = 2m - s_{k-1}$. Substituting $p_k = 2m - s_{k-1}$ into (3.5), we have $3m \leq \alpha\big(2m - s_{k-1} + \frac{s_{k-1}^2 - m s_{k-1} + m^2}{2m - s_{k-1}}\big)$, i.e.,

$$3 \leq \alpha \left(2 - \frac{s_{k-1}}{m} + \frac{(\frac{s_{k-1}}{m})^2 - \frac{s_{k-1}}{m} + 1}{2 - \frac{s_{k-1}}{m}} \right). \tag{3.6}$$

Note that the minimum value of the function

$$f_3\left(\frac{s_{k-1}}{m}\right) \doteq 2 - \frac{s_{k-1}}{m} + \frac{(\frac{s_{k-1}}{m})^2 - \frac{s_{k-1}}{m} + 1}{2 - \frac{s_{k-1}}{m}}$$

is achieved at $\frac{s_{k-1}}{m} = 2 - \sqrt{3/2}$, and $f_3(2 - \sqrt{3/2}) = 4\sqrt{3/2} - 3$. Since $\alpha \cdot (4\sqrt{3/2} - 3) = 3$, we are done.

Now we assume that $\frac{s_{k-1}}{m} \leq \frac{3}{5}$. The derivative of the left side of (3.5) with respect to s_{k-1} is $1 - \alpha\frac{2s_{k-1} - m}{p_k}$. If $1 - \alpha\frac{2s_{k-1} - m}{p_k} \leq 0$, then from $\frac{s_{k-1}}{m} \leq \frac{3}{5}$, we have $s_{k-1} + p_k \leq s_{k-1} + \alpha\,(2s_{k-1} - m) < \frac{3}{5}m + \frac{8}{5} \cdot \frac{1}{5}m < m$, which contradicts (3.3). In the opposite case this derivative is positive, thus we only need to show that (3.5) is valid when $s_{k-1} = \frac{3}{5}m$. That is to say,

$$p_k + \frac{3}{5}m + m \leq \alpha \left(p_k + \frac{\left(\frac{9}{25} - \frac{3}{5} + 1\right) m^2}{p_k} \right). \tag{3.7}$$

In fact, the right side of (3.7) equals $p_k + (\alpha - 1)p_k + \frac{19\alpha m^2}{25 p_k} \geq p_k + 2\sqrt{\frac{19(\alpha-1)\alpha}{25}}m > p_k + \frac{8}{5}m$, and thus the proof of Case 1 is finished.

Case 2. p_k is the first job assigned to the mth machine (i.e., p_k is assigned by step 3). Hence we have $m \geq 2$ and $s_{j,k-1} + p_k > 2(m-1)$ for all $1 \leq j \leq m-1$. From the algorithm rule, we know that $s_{j,k-1} \leq \max\{u_k, 2(m-1)\}$ for all $1 \leq j \leq m-1$. We obtain the following basic propositions which are useful in obtaining the desired competitive ratio for Case 2.

We first show that $u_k \leq 2(m-1)$. Otherwise, we suppose that $u_k > 2(m-1)$. Note that the makespan is no greater than u_k right before assigning p_k; we have $P_k = \sum_{j=1}^{m-1} s_{j,k-1} + p_k \leq (m-1)u_k + u_k = u_k m \leq u_k \cdot 2(m-1) < u_k^2$. It follows that $C'_k = C_{2,k}$. Furthermore, we get that $z_k \leq \max\{p_k, 2(m-1)\} + m \leq u_k + m < \frac{3}{2}(u_k + 1) \leq \frac{3}{2}\big(u_k + \frac{P_k}{u_k}\big)$, and we are done. Hence we have $u_k \leq 2(m-1)$. It follows that

$$C_{k-1} = \max\{s_{j,k-1} \mid 1 \leq j \leq m-1\} \leq 2(m-1). \tag{3.8}$$

Next we show that $P_k < (m - \frac{2}{3})^2$ and $p_k > m - \frac{2}{3}$. If $P_k \geq (m - \frac{2}{3})^2$, then we have $z_k \leq \max\{u_k, 2(m-1)\} + m \leq 3m - 2 \leq 3\sqrt{P_k} = \frac{3}{2}C_{1,k}$, and the desired competitive ratio is proved. Hence we assume that $P_k < (m - \frac{2}{3})^2$ in the following. If $p_k \leq m - \frac{2}{3}$, considering that $s_{j,k-1} > 2(m-1) - p_k$ for all $1 \leq j \leq m-1$, we have $P_k > (m-1)\,(2(m-1) - p_k) + p_k \geq 2(m-1)^2 - (m-2)\,\big(m - \frac{2}{3}\big) > (m - \frac{2}{3})^2$. Hence we can assume that $p_k > m - \frac{2}{3}$.

$P_k < (m - \frac{2}{3})^2$ and $p_k > m - \frac{2}{3}$, implying that $p_k \geq \sqrt{P_k}$; hence we have $C'_k = C_{2,k}$. Now we show the result by considering three subcases according to the values of C_{k-1} and p_k.

Subcase 2.1. $C_{k-1} \leq p_k$. In this subcase p_k is the largest job, and $z_k = p_k + m$. By Corollary 3.2 and Remark 3.1, we know that $P_k > \frac{3}{4}(m-1)m$; hence it suffices to prove $p_k + m \leq \frac{3}{2}\big(p_k + \frac{\frac{3}{4}(m-1)m}{p_k}\big)$, i.e.,

$$m \leq \frac{p_k}{2} + \frac{9(m-1)m}{8p_k}. \tag{3.9}$$

In fact, the right-hand side of (3.9) is at least $2\sqrt{\frac{9}{16}(m-1)m} = \frac{3}{2}\sqrt{(m-1)m} \geq m$; we thus finish Subcase 2.1.

Subcase 2.2. $C_{k-1} > p_k$ and $u_k \geq \frac{3}{2}(m-1)$. Similarly we have $P_k > \frac{3}{4}(m-1)^2 + m - 1$ by Corollary 3.2 and Remark 3.1. Combining it with (3.8), we only need to show that

$$z_k = C_{k-1} + m \leq 2(m-1) + m \leq \frac{3}{2}\left(u_k + \frac{\frac{3}{4}(m-1)^2 + m - 1}{u_k}\right). \tag{3.10}$$

Inequality (3.10) is equivalent to $0 \leq 12{u_k}^2 - 8(3m-2)u_k + 9m^2 - 6m - 3$, i.e.,

$$\left(u_k - \frac{3(m-1)}{2}\right)\left(u_k - \frac{3m+1}{6}\right) \geq 0. \tag{3.11}$$

Since $u_k \geq \frac{3}{2}(m-1)$, (3.11) is obviously true.

Subcase 2.3. $C_{k-1} > p_k$ and $u_k < \frac{3}{2}(m-1)$. For this subcase, we need to apply the induction assumption.

If $z_{k-1} \leq \alpha C_{1,k-1}$, i.e., $C_{k-1} + m - 1 \leq 2\alpha\sqrt{P_{k-1}}$, we show that $z_k \leq \alpha C_{1,k}$ also holds, i.e., $C_{k-1} + m \leq 2\alpha\sqrt{P_k} = 2\alpha\sqrt{P_{k-1} + p_k}$. It suffices to show that $2\alpha\sqrt{P_{k-1} + p_k} \geq 2\alpha\sqrt{P_{k-1}} + 1$, i.e.,

$$p_k \geq \frac{1}{\alpha}\sqrt{P_{k-1}} + \frac{1}{4\alpha^2}. \tag{3.12}$$

In fact, by applying $p_k > m - \frac{2}{3}$ and $P_k < (m - \frac{2}{3})^2$, we have $p_k > m - \frac{2}{3} = \frac{1}{\alpha}\left(m - \frac{2}{3}\right) + (1 - \frac{1}{\alpha})(m - \frac{2}{3}) > \frac{1}{\alpha}\sqrt{P_k} + \frac{1}{4\alpha^2} > \frac{1}{\alpha}\sqrt{P_{k-1}} + \frac{1}{4\alpha^2}$.

If $z_{k-1} \le \alpha C_{2,k-1}$, i.e., $C_{k-1} + m - 1 \le \alpha\big(u_{k-1} + \frac{P_{k-1}}{u_{k-1}}\big)$, then we have $u_{k-1} \ge \sqrt{P_{k-1}}$ by the induction assumption. It follows that $\alpha\big(u_{k-1} + \frac{P_{k-1}}{u_{k-1}}\big) \le \alpha\big(u_k + \frac{P_{k-1}}{u_k}\big)$. Hence we only need to show

$$\alpha\frac{P_{k-1} + p_k}{u_k} \ge \alpha\frac{P_{k-1}}{u_k} + 1 \tag{3.13}$$

to get $z_k = C_{k-1} + m \le \alpha\big(u_k + \frac{P_{k-1}+p_k}{u_k}\big) = \alpha C_{2,k}$. Inequality (3.13) is equivalent to $p_k \ge \frac{1}{\alpha}u_k$, which immediately follows from $p_k > m - \frac{2}{3}$ and $u_k < \frac{3}{2}(m-1)$.

In summary, we have shown that the result is true for $l = k$. We thus get Theorem 3.4. □

3.2. Optimal online algorithm for small job case. In this subsection we assume that all jobs have sizes no greater than 1, i.e., $L \le 1$. We present an online algorithm which is optimal with a competitive ratio 4/3. Note that by considering the job sequence $p_1 = \cdots = p_{4N} = 1/N$, where N is a sufficiently large positive integer, we get $A_\rho(I)/OPT(I) = (6-1/N)/4 \longrightarrow 3/2$. It concludes that the competitive ratio of A_ρ cannot be smaller than 3/2 for small job case.

ALGORITHM $H2$. The algorithm works in the same way as $H1$ except that step 2 is defined as follows:

2. Assign p_k to the machine with minimum current load if the new load will be no more than $m+1$, and go to 4. Else go to 3.

LEMMA 3.5. *Let $m \ge 2$ be an integer, and let b satisfy $m < b \le m+1$.*

(1) *Define the function $f_4(i) = \frac{(m-1)^2+1}{i} + i$ for any integer $i \ge 1$. Then the minimum value of $f_4(i)$ is $2(m-1) + \frac{1}{m-1}$.*

(2) *Define the function $f_5(i) = \frac{m(b-1)}{i} + i$ for any integer $i \ge 1$. Then the minimum value of $f_5(i)$ is $b + m - 1$.*

Proof. (1) Define $g(x) = \frac{(m-1)^2+1}{x} + x$ for any real number $x \ge 1$. Then

$$g'(x) = \frac{1}{x^2}\left(x^2 - \left((m-1)^2+1\right)\right) = \frac{x + \sqrt{(m-1)^2+1}}{x^2}\left(x - \sqrt{(m-1)^2+1}\right),$$

$$g''(x) = \frac{2((m-1)^2+1)}{x^3} > 0.$$

Therefore the function $g(x)$ achieves its minimum value at $x = \sqrt{(m-1)^2+1}$, and the function $f_4(i)$ achieves its minimum at $i = \left\lfloor\sqrt{(m-1)^2+1}\right\rfloor = m-1$ or $i = \left\lceil\sqrt{(m-1)^2+1}\right\rceil = m$. On the other hand, it is easy to see that $f_4(m-1) = \frac{(m-1)^2+1}{m-1} + m - 1 \le f_4(m) = \frac{(m-1)^2+1}{m} + m$ for any given $m \ge 2$. We thus get (1).

(2) It can be proved similarly that the minimum value is achieved at $i = m$. □

THEOREM 3.6. *The competitive ratio of $H2$ is 4/3 if every job has size no greater than 1, and thus it is optimal.*

Proof. Analogous to the proof of Theorem 3.4, we show the result by distinguishing the value of m', the number of machines purchased by $H2$ right after scheduling all jobs, and the assignment of the last job p_n.

Case 1. $m' = 1$. Then by the algorithm rule, we have $C_{\max} = P = \sum_{i=1}^{n} p_i \le 2$, and $H2(I) = C_{\max} + m' \le 3$. It follows that there are at most two machines in the

optimal schedule. If there is one machine in the optimal schedule, then obviously $H2$ yields the optimal schedule. Otherwise, $OPT(I) \geq \frac{P}{2} + 2$ by averaging argument. It is straightforward to see that $H2(I) = P + 1 \leq \frac{4}{3}\left(\frac{P}{2} + 2\right) \leq \frac{4}{3}OPT(I)$.

Case 2. $m' \geq 2$ and p_n is the first job assigned to the m'th machine (i.e., p_n is assigned by step 3). Then right before scheduling p_n, there are exactly $m'-1$ machines. By the algorithm rule, we have $s_{j,n-1} + p_n > m'$ for every $j = 1, \ldots, m'-1$. Thus we get

$$\sum_{j=1}^{m'-1} (s_{j,n-1} + p_n) = \sum_{j=1}^{m'-1} s_{j,n-1} + (m'-1)p_n > m'(m'-1).$$

Combining it with $p_n \leq 1$, we obtain

$$\begin{aligned} P &= \sum_{j=1}^{m'-1} s_{j,n-1} + p_n > m'(m'-1) - (m'-2)\, p_n \\ &\geq m'(m'-1) - (m'-2) = (m'-1)^2 + 1. \end{aligned}$$

By averaging argument, we have $OPT(I) \geq \min_{i \geq 1}\left\{\frac{(m'-1)^2+1}{i} + i\right\}$, where i is the possible machine number in the optimal schedule. By Lemma 3.5(1), we get that $OPT(I) \geq 2(m'-1) + \frac{1}{m'-1}$. On the other hand, it is clear that $C_{\max} \leq m'$ from the algorithm rule; thus $H2(I) = C_{\max} + m' \leq 2m'$. For every $m' \geq 2$, $\frac{2m'-3}{2m'^2-4m'+3} \leq \frac{4}{3}$ holds trivially. Therefore we have

$$\frac{H2(I)}{OPT(I)} \leq \frac{2m'}{2(m'-1) + \frac{1}{m'-1}} = \frac{2\,(m'-1)\,m'}{2(m'-1)^2+1} = 1 + \frac{2m'-3}{2m'^2-4m'+3} \leq \frac{4}{3}.$$

Case 3. $m' \geq 2$ and p_n is assigned to one of m' machines by step 2. It is clear that the final makespan $C_{\max} \leq m'+1$ and $H2(I) = C_{\max} + m'$. If further $C_{\max} \leq m'$, we have $H2(I) \leq 2m'$. Assume that the m'th machine is purchased when assigning job p_k. By the same argument as that in Case 2, we get $P_n \geq P_k \geq (m'-1)^2 + 1$, and thus $H2(I)/OPT(I) \leq 4/3$.

Hence we can assume that $m' < C_{\max} \leq m'+1$. Since p_n is assigned to the machine with minimum current load, the load of every machine is at least $C_{\max} - p_n \geq C_{\max} - 1$; thus the sum of all sizes $P \geq (m'-1)\,(C_{\max}-1) + C_{\max} > m'\,(C_{\max}-1)$. It follows that $OPT(I) \geq \min_{i \geq 1}\left\{\frac{m'(C_{\max}-1)}{i} + i\right\}$. By Lemma 3.5(2), we get $OPT(I) \geq C_{\max} + m' - 1$. Hence we get that

$$\frac{H2(I)}{OPT(I)} \leq \frac{C_{\max} + m'}{C_{\max} + m' - 1} = 1 + \frac{1}{C_{\max} + m' - 1} \leq \frac{4}{3},$$

where the last inequality follows from $m' \geq 2$ and $C_{\max} > m' \geq 2$.

Therefore we have shown that $H2$ is 4/3-competitive, and the optimality is obtained directly from Lemma 2.3. □

Remark 3.2. $H2$ also works for the case $1 < L \leq 3$ by simple modification as follows: The algorithm is the same as $H2$ except that in step 2, $m+1$ is replaced by $m + L$. It can be shown similarly that the competitive ratio of modified $H2$ is no greater than 3/2 for the problem where L is known in advance and $1 < L \leq 3$.

4. Semionline algorithm for the problem with known largest size. In this section, we assume that jobs arrive one by one and the largest size L of all jobs is known in advance. As we have seen in previous sections, if $L \le 1$, $H2$ is 4/3-competitive, and if $1 < L \le 3$, both A_ρ and modified $H2$ have a competitive ratio at most 3/2; therefore we assume that $L > 3$ in the remainder of the section. We present a two-phase algorithm with a competitive ratio at most 3/2, which uses new strategies to purchase machines in two phases.

ALGORITHM $H3$.

PHASE 1.

1.1 Let $k = 1$, $m = 1$.

1.2 Assign p_k to the machine with minimum current load if the new load will be no more than $\frac{3}{2}L$, and go to 1.4. Else go to 1.3.

1.3 Assign p_k to a new machine, and let $m = m + 1$. If $m = m_0$, then go to 2.6 in Phase 2, else go to 1.4.

1.4 Let $k = k + 1$, if $k > n$, stop. Otherwise, go to 1.2.

PHASE 2.

2.1 Compute the current value $C_{1,k} = 2\sqrt{P_k}$, and the minimum current machine load s_{k-1}.

2.2 If $s_{k-1} + p_k$ is not greater than the current makespan, then assign p_k to the machine with minimum current load, go to 2.6.

2.3 If $P_k > m^2$, then assign p_k to a new machine, let $m = m + 1$, go to 2.6.

2.4 If $s_{k-1} + p_k + m > \frac{3}{2}C_{1,k}$, then assign p_k to a new machine, let $m = m + 1$, go to 2.6.

2.5 Else assign p_k to the machine with minimum current load, go to 2.6.

2.6 Let $k = k + 1$. If $k > n$, stop. Else return 2.1.

THEOREM 4.1. *The competitive ratio of $H3$ is at most 3/2.*

We prove Theorem 4.1 step by step by showing a series of technical lemmas in the following. We first consider Phase 1.

LEMMA 4.2. *For every $m \le m_0$, we have $P_{i_m} > \frac{3}{4}mL$.*

Proof. It is clear that $P_{i_m} = \sum_{l=1}^{m} s_{l,i_m}$. By the rule of Phase 1, we know that $s_{l,i_m} + s_{j,i_m} \ge s_{i_m} + p_{i_j} > \frac{3}{2}L$ for all $1 \le l < j \le m$. Summing up these inequalities, we get $P_{i_m} > \frac{3}{4}mL$. □

LEMMA 4.3. *If $H3$ stops at Phase 1, then the competitive ratio is no greater than 3/2.*

Proof. Suppose a total of $m' \le m_0$ machines are purchased by $H3$. Then by Lemma 4.2, we know $P \ge P_{i_{m'}} > \frac{3}{4}m'L$. If $P \le L^2$, then by Lemma 2.1 we get

$$\begin{aligned} \frac{3}{2}OPT(I) \ge \frac{3}{2}C_{2,n} &= \frac{3}{2}\left(L + \frac{P}{L}\right) > \frac{3}{2}\left(L + \frac{3}{4}m'\right) \\ &= \frac{3}{2}L + \frac{9}{8}m' > C_{\max} + m' = H3(I). \end{aligned} \tag{4.1}$$

If $P > L^2$, then again by Lemma 2.1 we have

$$\frac{3}{2}OPT(I) \ge \frac{3}{2}C_{1,n} \ge 3\sqrt{L^2} = \frac{3}{2}L + \frac{3}{2}L \ge C_{\max} + m' = H3(I), \tag{4.2}$$

where the last inequality follows from $m' \le m_0 = \lceil L \rceil$ and $L > 3$. □

In the remainder of this section, we focus on analyzing Phase 2. We are going to prove $z_k \le \frac{3}{2}C_{1,k}$ for every $k = i_{m_0}, \ldots, n$, from which it follows that the competitive ratio of $H3$ is at most 3/2. Note that at the beginning of Phase 2 we have purchased

exactly m_0 machines, there is only one job $p_{i_{m_0}}$ assigned to the m_0th machine, the total size of all arrived jobs Q_{m_0} is between $\frac{3}{4}m_0L$ and $\frac{3}{2}m_0L$, and the current makespan is no greater than $\frac{3}{2}L$.

LEMMA 4.4. $z_{i_{m_0}} \le \frac{3}{2}C_{1,i_{m_0}}$.

Proof. By Lemma 4.2, we get $P_{i_{m_0}} \ge \frac{3}{4}m_0L$. As $z_{i_{m_0}} \le m_0 + \frac{3}{2}L$ and $C_{1,i_{m_0}} = 2\sqrt{P_{i_{m_0}}}$, it suffices to show $3\sqrt{\frac{3}{4}m_0L} \ge \frac{3}{2}L + m_0$, i.e., $9\left(\frac{L}{m_0}\right)^2 - 15\left(\frac{L}{m_0}\right) + 4 \le 0$. Since $L > 3$, it is obvious that $\frac{1}{3} \le \frac{L}{m_0} \le \frac{4}{3}$, from which it follows that the desired inequality is true. ∎

Notice that in Phase 2, the competitive ratio may become worse than 3/2 only when a new machine is purchased by steps 2.3 and 2.4 and the incoming job is assigned to this machine. Thus if $z_{k-1} \le \frac{3}{2}C_{1,k-1}$ holds and job p_k is assigned by step 2.2 or by step 2.5, then we have $z_k \le \frac{3}{2}C_{1,k}$. In the following, we mainly focus on analyzing steps 2.3 and 2.4, that is, considering the assignment of p_{i_m} and estimating z_{i_m}, $m \ge m_0 + 1$.

LEMMA 4.5. *At Phase 2, if $z_k \le \frac{3}{2}C_{1,k}$ holds, then we know that the current makespan $C_k \le 2m$, where m is the number of currently purchased machines.*

Proof. It is clear that no job can have a size greater than $2m$ at Phase 2. Hence the makespan may be increased to more than $2m$ only at step 2.5. But in this case we have $P_k \le m^2$, and thus $z_k = C_k + m \le \frac{3}{2}C_{1,k} \le 3m$. It follows that $C_k \le 2m$. ∎

LEMMA 4.6. *If the mth machine is purchased by step 2.3, where $m \ge m_0 + 1$, and if $z_{i_m-1} \le \frac{3}{2}C_{1,i_m-1}$ holds, then we have $z_{i_m} \le \frac{3}{2}C_{1,i_m}$.*

Proof. Since the mth machine is purchased by step 2.3 (i.e., p_{i_m} is not assigned by step 2.2), there exists some $\beta > 1$ such that $P_{i_m} = \beta(m-1)^2$, and

$$C_{i_m-1} < s_{i_m-1} + p_{i_m} \le s_{i_m-1} + L \le s_{i_m-1} + m - 1. \tag{4.3}$$

By Lemma 4.5, we get that

$$P_{i_m} \le (m-1)C_{i_m-1} + p_{i_m} \le 2(m-1)^2 + (m-1) \le 3(m-1)^2, \tag{4.4}$$

and thus $\beta \le 3$. We estimate the value of C_{i_m-1} as follows. If $C_{i_m-1} > 3\sqrt{\beta}(m-1) - m$, then we know from (4.3) that $s_{i_m-1} > 3\sqrt{\beta}(m-1) - m - (m-1)$. Therefore

$$\begin{aligned} P_{i_m} &= \sum_{j=1}^{m-1} s_{j,i_m-1} + p_{i_m} \ge (m-2)\, s_{i_m-1} + C_{i_m-1} \\ &> (m-1)\left(3\sqrt{\beta}(m-1) - m\right) - (m-2)\,(m-1) \\ &= (m-1)^2(3\sqrt{\beta} - 2) > \beta(m-1)^2 = P_{i_m}, \end{aligned} \tag{4.5}$$

where the last inequality follows from $1 < \beta \le 3$—a contradiction. Hence we have $C_{i_m-1} \le 3\sqrt{\beta}(m-1) - m$. With this estimation, we obtain

$$z_{i_m} = C_{i_m-1} + m \le 3\sqrt{\beta}(m-1) = 3\sqrt{P_{i_m}} = \frac{3}{2}C_{1,i_m},$$

or

$$z_{i_m} = p_{i_m} + m \le 2m - 1 < 3(m-1) < 3\sqrt{\beta}(m-1) = \frac{3}{2}C_{1,i_m}. \quad ∎$$

LEMMA 4.7. *If the $(m-1)$th machine is purchased by step 2.3, where $m > m_0+1$, then so is the mth machine. It follows that $H3$ will not enter step 2.4 since then, i.e., all remaining new machines (if necessary) are purchased by step 2.3.*

Proof. If $P_{i_m} > (m-1)^2$, i.e., all the arrived jobs have total size greater than $(m-1)^2$ before purchasing the mth machine, then the algorithm enters step 2.3 by the rule, and the statement is clearly true. Hence we assume that $P_{i_m-1} \le P_{i_m} \le (m-1)^2$. Since the $(m-1)$th machine is purchased by step 2.3, we get $P_{i_m-1} \ge P_{i_{m-1}} > (m-2)^2$ by the algorithm rule.

To prove that the mth machine must be purchased by step 2.3, we verify that $s_{i_m-1} + p_{i_m} + m - 1 \le \frac{3}{2} C_{1,i_m}$. To see it, since $s_{i_m-1} \le \frac{P_{i_m-1}}{m-1}$ by averaging argument and $C_{1,i_m} = 2\sqrt{P_{i_m-1} + p_{i_m}}$, it suffices to verify that

$$\frac{P_{i_m-1}}{m-1} + p_{i_m} + m - 1 - 3\sqrt{P_{i_m-1} + p_{i_m}} \le 0. \tag{4.6}$$

The derivative of the left side of (4.6) with respect to P_{i_m-1} is $\frac{1}{m-1} - \frac{3}{2\sqrt{P_{i_m-1}+p_{i_m}}}$. From $P_{i_m-1} \le (m-1)^2$ and $p_{i_m} \le L \le m-2$, we know that the derivative is negative. Thus from $P_{i_m-1} > (m-2)^2$, it suffices to show that

$$\frac{(m-2)^2}{m-1} + p_{i_m} + m - 1 - 3\sqrt{(m-2)^2 + p_{i_m}} \le 0, \tag{4.7}$$

i.e.,

$$2m - 4 + \frac{1}{m-1} + p_{i_m} - 3\sqrt{(m-2)^2 + p_{i_m}} \le 0. \tag{4.8}$$

The derivative of the left side of (4.8) with respect to p_{i_m} is $1 - \frac{3}{2\sqrt{(m-2)^2+p_{i_m}}} > 0$ because $m > m_0 + 1 \ge 5$. Since $p_{i_m} \le m-2$, $\frac{1}{m-1} < 1$, and $3m - 5 - 3\sqrt{(m-2)^2 + m - 2} \le 0$, we can conclude that (4.8) is true. The lemma is thus proved. □

Remark 4.1. From Lemmas 4.6 and 4.7, we can conclude that once a new machine is purchased by step 2.3—and at this time the competitive ratio of $H3$ is no greater than 3/2—then the competitive ratio cannot become worse than 3/2 after that. For example, if the (m_0+1)th machine is purchased by step 2.3, then from Lemma 4.4, we conclude that the competitive ratio of $H3$ must be no greater than 3/2 after assigning all jobs. Take another example: if there exists some $m \ge m_0 + 1$ such that $P_{i_m} > (m-1)^2$, and the competitive ratio is not greater than 3/2 before purchasing the mth machine, then we can also conclude that the competitive ratio of $H3$ must be no greater than 3/2 after assigning all jobs. Hence we assume that $P_{i_m} \le (m-1)^2$ for every $m \ge m_0 + 1$.

LEMMA 4.8. $s_{i_{m_0+1}-1} > \frac{m_0}{2}$.

Proof. Otherwise $s_{i_{m_0+1}-1} + p_{i_{m_0+1}} + m_0 \le \frac{3}{2} m_0 + L$. Analogous to the proof of Lemma 4.4, we get that $\frac{3}{2} m_0 + L \le 3\sqrt{\frac{3}{4} L m_0} \le \frac{3}{2} C_{1,s_{i_{m_0+1}}}$. Then it follows that the (m_0+1)th machine is purchased by step 2.3, and by Remark 4.1 the proof is complete. □

LEMMA 4.9. *If $m_0 \le 11$ and $P_{i_{m_0+1}} \le m_0^2$, then $s_{i_{m_0+1}-1} + p_{i_{m_0+1}} + m_0 \le \frac{3}{2} C_{1,i_{m_0+1}}$. It follows that the (m_0+1)th machine is purchased by step* 2.3.

Proof. Otherwise, we have

$$s_{i_{m_0+1}-1} + p_{i_{m_0+1}} + m_0 > \frac{3}{2} C_{1,i_{m_0+1}} = 3\sqrt{P_{i_{m_0+1}}}, \tag{4.9}$$

and the (m_0+1)th machine would be purchased by step 2.4. We distinguish two cases according to the value of $s_{i_{m_0+1}-1}$ to get a contradiction.

Case 1. $s_{i_{m_0+1}-1} \geq \frac{3}{4}L$. Because $P_{i_{m_0+1}} \geq s_{i_{m_0+1}-1}m_0 + p_{i_{m_0+1}}$, (4.9) implies that

$$s_{i_{m_0+1}-1} + p_{i_{m_0+1}} + m_0 - 3\sqrt{s_{i_{m_0+1}-1}m_0 + p_{i_{m_0+1}}} > 0. \tag{4.10}$$

The derivative of the left side of (4.10) with respect to $p_{i_{m_0+1}}$ is

$$1 - \frac{3}{2\sqrt{s_{i_{m_0+1}-1}m_0 + p_{i_{m_0+1}}}} > 0$$

because $m_0 \geq L > 3$ and $s_{i_{m_0+1}-1} \geq \frac{3}{4}L$. Thus it follows from $p_{i_{m_0+1}} \leq L$ that

$$s_{i_{m_0+1}-1} + L + m_0 - 3\sqrt{s_{i_{m_0+1}-1}m_0 + L} > 0. \tag{4.11}$$

Again, the derivative of the left side of (4.11) with respect to $s_{i_{m_0+1}-1}$ is

$$1 - \frac{3m_0}{2\sqrt{s_{i_{m_0+1}-1}m_0 + L}}.$$

$P_{i_{m_0+1}} \leq m_0^2$ implies $s_{i_{m_0+1}-1} \leq m_0$. Combining it with $s_{i_{m_0+1}-1}m_0 + L \leq m_0^2 + m_0 < \frac{9}{4}m_0^2$, we know that the derivative is negative. Because $s_{i_{m_0+1}-1} \geq \frac{3}{4}L$, it follows from (4.11) that

$$\frac{3}{4}L + L + m_0 > 3\sqrt{\frac{3}{4}Lm_0 + L}, \tag{4.12}$$

i.e., $(\frac{7}{4}L + m_0)^2 > (3\sqrt{\frac{3}{4}Lm_0 + L})^2$. Therefore we have

$$f_6(L) \doteq 49L^2 - (52m_0 + 144)\,L + 16m_0^2 > 0 \tag{4.13}$$

for any $L > 3$ and $m_0 = \lceil L \rceil \geq 4$.

On the other hand, it is clear that the function $f_6(L)$ achieves the maximum value at $L = m_0$ if $m_0 > 4$, and at $L = m_0 - 1$ if $m_0 = 4$. Therefore the maximum value is $f_6(3) = 49 \cdot 3^2 - (52 \cdot 4 + 144)\,3 + 16 \cdot 4^2 = -359 < 0$ if $m_0 = 4$, or $f_6(m_0) = 49m_0^2 - (52m_0 + 144)\,m_0 + 16m_0^2 < 0$ if $4 < m_0 \leq 11$, which contradicts (4.13).

Case 2. $s_{i_{m_0+1}-1} < \frac{3}{4}L$. Then from (4.9) and Lemma 4.2, we get

$$s_{i_{m_0+1}-1} + p_{i_{m_0+1}} + m_0 - 3\sqrt{\frac{3}{4}Lm_0 + p_{i_{m_0+1}}} > 0. \tag{4.14}$$

Again, the derivative of the left side with respect to $p_{i_{m_0}+1}$ is positive; thus it follows from $p_{i_{m_0}+1} \leq L$ that

$$s_{i_{m_0+1}-1} + L + m_0 - 3\sqrt{\frac{3}{4}Lm_0 + L} > 0. \tag{4.15}$$

Substituting $s_{i_{m_0+1}-1} < \frac{3}{4}L$ into (4.15), we get (4.11) again. Then the same argument can yield a contradiction. □

By Lemma 4.9, we know that $H3$ may enter step 2.4 only if $m_0 \geq 12$ (and $P_{i_m} \leq (m-1)^2$ for every $m \geq m_0+1$, as declared before). Hence we further assume in the following that $m_0 \geq 12$. Suppose that $H3$ is worse than 3/2-competitive. Then there must exist an m with $m_0+1 \leq m$ such that the mth machine is purchased by step 2.4, and right after assigning p_{i_m} to this machine, $H3$ becomes worse than 3/2-competitive. In the following lemmas we will see that there is no such m. Next Lemmas 4.10–4.12 consider the case $m_0 \leq m \leq 2m_0$, and Lemmas 4.13 and 4.14 consider the case $m > 2m_0$.

LEMMA 4.10. *For every $m_0 \leq m \leq 2m_0$, we have $p_{i_{m+1}} > s_{i_{m+1}-1}$.*

Proof. First we show that the result is true for $m = m_0$. Suppose in contrast that $p_{i_{m_0+1}} \leq s_{i_{m_0+1}-1}$. We show that $s_{i_{m_0+1}-1} + p_{i_{m_0+1}} + m_0 \leq 3\sqrt{P_{i_{m_0+1}}}$, which implies that the (m_0+1)th machine is purchased by step 2.3. It is clear that $P_{i_{m_0+1}} \geq m_0 s_{i_{m_0+1}-1}$. Thus it suffices to prove that $2s_{i_{m_0+1}-1} + m_0 \leq 3\sqrt{m_0 s_{i_{m_0+1}-1}}$. This is equivalent to $\frac{1}{4} \leq \frac{s_{i_{m_0+1}-1}}{m_0} \leq 1$, which is obviously true since $s_{i_{m_0+1}-1} > \frac{m_0}{2}$ by Lemma 4.8 and $s_{i_{m_0+1}-1} \leq \frac{P_{i_{m_0+1}-1}}{m_0} \leq m_0$. Hence we get $p_{i_{m_0+1}} > s_{i_{m_0+1}-1}$.

Since the minimum current machine load is not decreased before or right when a new machine is purchased, hence we know that $s_{i_{m_0+2}-1} \geq \min\{p_{i_{m_0+1}}, s_{i_{m_0+1}-1}\} > \frac{m_0}{2}$. On the other hand, $s_{i_{m_0+2}-1} \leq \frac{P_{i_{m_0+2}-1}}{m_0+1} \leq m_0+1$, and hence we have $\frac{1}{4} \leq \frac{s_{i_{m_0+2}-1}}{m_0+1} \leq 1$. Then by the same argument as above we can show that $p_{i_{m_0+2}} > s_{i_{m_0+2}-1}$. The result for other $m = m_0+2, \ldots, 2m_0$ can be obtained similarly. ∎

Remark 4.2. By Lemma 4.10, we know that through $m_0 \leq m \leq 2m_0$, the minimum current machine load is not decreased. That is, $s_{i_{m+1}-1} \geq s_{i_{m_0+1}-1}$ for any $m_0+1 \leq m \leq 2m_0$. Furthermore if $s_{i_{m_0+1}-1} \geq \frac{3}{4}L$, then $p_{i_{m+1}} \geq s_{i_{m_0+1}-1} \geq \frac{3}{4}L$ for all $m_0 \leq m \leq 2m_0$, and thus from Lemma 4.2, we get $P_{i_m} \geq P_{i_{m_0}} + p_{i_{m_0+1}} + \cdots + p_{i_m} \geq \frac{3}{4}Lm_0 + \frac{3}{4}L\,(m-m_0) = \frac{3}{4}Lm$ for every $m_0 \leq m \leq 2m_0$.

LEMMA 4.11. *For every $m_0 \leq m \leq 2m_0$, we have $p_{i_{m+1}} \geq \frac{3}{4}L$, and thus $P_{i_m} \geq \frac{3}{4}Lm$.*

Proof. Suppose $p_{i_{m_0+1}} < \frac{3}{4}L$. By Remark 4.2, we only need to consider the case $s_{i_{m_0+1}-1} < \frac{3}{4}L$. It suffices to show that $s_{i_{m_0+1}-1} + p_{i_{m_0+1}} + m_0 \leq 3\sqrt{P_{i_{m_0+1}}}$, which implies that the (m_0+1)th machine is purchased by step 2.3, and we are done by Remark 4.1. Because of $s_{i_{m_0+1}-1} < \frac{3}{4}L$, we only need to show that $\frac{3}{2}L + m_0 \leq 3\sqrt{\frac{3}{4}Lm_0}$. This is equivalent to $\frac{1}{3} \leq \frac{L}{m_0} \leq \frac{4}{3}$, which is obviously true. Hence we have $p_{i_{m_0+1}} \geq \frac{3}{4}L$ and thus $P_{i_{m_0+1}} \geq \frac{3}{4}L(m_0+1)$ by an argument similar to that in Remark 4.2. The result for $m = m_0+1, \ldots, 2m_0$ can be obtained similarly one by one. ∎

LEMMA 4.12. *If for some $m_0 \leq m \leq 2m_0$, the $(m+1)$th machine is purchased by step 2.4, and $z_{i_{m+1}-1} \leq \frac{3}{2}C_{1,i_{m+1}-1}$ holds, then we have $z_{i_{m+1}} \leq \frac{3}{2}C_{1,i_{m+1}}$.*

Proof. If $p_{i_{m+1}} \geq \max\{s_{j,i_{m+1}-1} |\ j = 1, \ldots, m\}$, then it is easy to see that $z_{i_{m+1}} = p_{i_{m+1}} + m + 1 \leq L + m + 1 \leq 3\sqrt{\frac{3}{4}Lm} \leq \frac{3}{2}C_{1,i_{m+1}}$, where the last inequality follows from Lemma 4.11. In the opposite case the objective value is increased by 1, because the makespan remains unchanged. Since $z_{i_{m+1}-1} \leq \frac{3}{2}C_{1,i_{m+1}-1}$ holds, to get the desired result, it suffices to show that $3\sqrt{P_{i_{m+1}-1} + p_{i_{m+1}}} \geq 3\sqrt{P_{i_{m+1}-1}} + 1$, i.e.,

$$p_{i_{m+1}} \geq \frac{2}{3}\sqrt{P_{i_{m+1}-1}} + \frac{1}{9}. \tag{4.16}$$

Let $P_{i_{m+1}-1} = \beta m^2$ with some $\beta \le 1$. From $\frac{L}{m} > \frac{m_0-1}{2m_0} \ge \frac{1}{2} - \frac{1}{24} > \frac{4}{9}$ and Lemma 4.11, we get $P_{i_{m+1}-1} \ge P_{i_m} \ge \frac{3}{4}Lm > \frac{1}{3}m^2$. Thus $\frac{1}{3} < \beta \le 1$. Since the $(m+1)$th machine is purchased by step 2.4, we have

$$s_{i_{m+1}-1} + p_{i_{m+1}} + m > 3\sqrt{P_{i_{m+1}-1}} = 3\sqrt{\beta}m. \tag{4.17}$$

By applying $s_{i_{m+1}-1} \le \frac{P_{i_{m+1}-1}}{m} = \beta m$, we get $p_{i_{m+1}} > \left(3\sqrt{\beta} - \beta - 1\right) m$. Thus (4.16) holds if we can show that

$$\left(3\sqrt{\beta} - \beta - 1\right) m > \frac{2}{3}\sqrt{\beta}m + \frac{1}{9}, \tag{4.18}$$

i.e., $\left(\frac{7}{3}\sqrt{\beta} - \beta - 1\right) m > \frac{1}{9}$. This inequality holds obviously since $\frac{1}{3} < \beta \le 1$ and $m \ge 12$. □

By Lemmas 4.10–4.12, we conclude that the competitive ratio of Algorithm $H3$ must be no greater than 3/2 before the $(2m_0+1)$th machine is purchased.

LEMMA 4.13. *Let m satisfy $m_0 \le m \le 2m_0$. Then we know $p_{i_{m+1}} > y$, where*

$$y = 3\sqrt{\frac{3}{4}Lm_0} - m_0 - \frac{3}{4}L.$$

Proof. By the algorithm rule it is clear that for any $m_0 \le m \le 2m_0$, we have $s_{i_{m+1}-1} + p_{i_{m+1}} + m > \frac{3}{2}C_{1,i_{m+1}} \ge 3\sqrt{s_{i_{m+1}-1}m}$, since otherwise the mth machine is purchased by step 2.3. Hence we obtain

$$p_{i_{m+1}} > 3\sqrt{s_{i_{m+1}-1}m} - s_{i_{m+1}-1} - m. \tag{4.19}$$

Define the function $f(s_t, m) = 3\sqrt{s_t m} - s_t - m$. It is clear that this function is symmetric, and increasing in both variables, since now $s_t < m$ holds. Recall that for any $m_0 \le m_1 < m_2 \le 2m_0$, we have that $s_{i_{m_1}} \le s_{i_{m_2}}$ by Remark 4.2.

If $s_{i_{m+1}-1} \ge \frac{3}{4}L$, we then have

$$\begin{aligned} p_{i_{m+1}} &> 3\sqrt{s_{i_{m+1}-1}m} - s_{i_{m+1}-1} - m \\ &\ge 3\sqrt{s_{i_{m+1}-1}m_0} - s_{i_{m+1}-1} - m_0 \\ &\ge 3\sqrt{s_{i_{m_0+1}-1}m_0} - s_{i_{m_0+1}-1} - m_0 \\ &\ge 3\sqrt{\frac{3}{4}Lm_0} - \frac{3}{4}L - m_0 = y. \end{aligned} \tag{4.20}$$

If $s_{i_{m+1}-1} < \frac{3}{4}L$, then by Lemma 4.11 and the algorithm rule, we have $p_{i_{m+1}} + s_{i_{m+1}-1} + m > \frac{3}{2}C_{1,i_{m+1}} \ge 3\sqrt{\frac{3}{4}Lm}$. Thus we get similarly that

$$\begin{aligned} p_{i_{m+1}} &> 3\sqrt{\frac{3}{4}Lm} - s_{i_{m+1}-1} - m \\ &> 3\sqrt{\frac{3}{4}Lm} - \frac{3}{4}L - m \\ &\ge 3\sqrt{\frac{3}{4}Lm_0} - \frac{3}{4}L - m_0 = y. \end{aligned} \tag{4.21}$$

□

LEMMA 4.14. *The* $(2m_0+1)$*th machine is not purchased by step* 2.4.

Proof. First we consider the case $s_{i_{2m_0+1}-1} \geq \frac{3}{4}L$. By Lemma 4.13, we get that

$$\begin{aligned} P_{i_{2m_0+1}} &\geq m_0 s_{i_{2m_0+1}-1} + p_{i_{m_0+1}} + \cdots + p_{i_{2m_0}} \\ &> m_0 s_{i_{2m_0+1}-1} + m_0 y \\ &= m_0 \left(s_{i_{2m_0+1}-1} + y \right), \end{aligned}$$

where $y = 3\sqrt{\frac{3}{4}Lm_0} - \frac{3}{4}L - m_0$. To get the result, due to $p_{i_{2m_0+1}} \leq L$, we only need to show that

$$L + s_{i_{2m_0+1}-1} + 2m_0 < 3\sqrt{m_0 \left(s_{i_{2m_0+1}-1} + y \right)}. \tag{4.22}$$

By an argument similar to what we have used before, it suffices to prove

$$\frac{7}{4}L + 2m_0 < 3\sqrt{3m_0\sqrt{\frac{3}{4}Lm_0} - m_0^2}, \tag{4.23}$$

which can be verified easily since $L \leq m_0$ and $m_0 \geq 12$.

For the case $s_{i_{2m_0+1}-1} < \frac{3}{4}L$, we have $P_{i_{2m_0+1}} \geq P_{i_{2m_0}} \geq \frac{3}{4}Lm_0 + m_0 y = 3m_0\sqrt{\frac{3}{4}Lm_0} - m_0^2$. Hence we need to show that

$$L + s_{i_{2m_0+1}-1} + 2m_0 < 3\sqrt{3m_0\sqrt{\frac{3}{4}Lm_0} - m_0^2}. \tag{4.24}$$

Since $s_{i_{2m_0+1}-1} < \frac{3}{4}L$, it suffices to prove the stronger inequality (4.23), which can be done by the same argument. □

Combining Lemmas 4.14 and 4.12 and Remark 4.1, we have shown in all cases that Algorithm $H3$ has a competitive ratio no greater than 3/2. We thus finish the proof of Theorem 4.1.

5. Conclusions. Even for the basic online scheduling problem $Pm|online|C_{\max}$, where the machine number m is fixed, we do not know its optimal online algorithm for any $m > 3$; therefore it may not be easy to get the optimal online algorithm for the list model problem, too. In this paper, we first devised an online algorithm with a competitive ratio at most $(2\sqrt{6}+3)/5 \approx 1.5798$. It improves upon the known algorithm, which has a competitive ratio $(1+\sqrt{5})/2 \approx 1.618$. As another attempt to make problem easier to solve, in this paper we considered a special case with small jobs. We presented its optimal online algorithm with a competitive ratio 4/3. We further considered the semionline problem with known largest processing time. A new algorithm with a competitive ratio no greater than 3/2 was presented which improves the known upper bound 1.5309.

To tighten the upper bounds, our algorithms applied new strategies for deciding when to purchase machines. Their decision regarding when to purchase machines depends on the current makespan, and upon the current objective value with respect to the lower bounds of the current optimal value, which is quite different from A_ρ, whose decision simply depends on the total size of arrived jobs. We conjecture that it is valuable to get a tighter lower bound of the optimal value for improving online and semionline algorithms (with competitive analysis). On the other hand, all known algorithms assign jobs to machines in greedy ways (by the classical LS rule). It is well

known that LS is not an optimal algorithm for the problem $Pm|online|C_{\max}$ when $m > 3$. Hence it is interesting whether we can obtain better algorithms by combining the new purchasing strategies and improved algorithms for $Pm|online|C_{\max}$.

Although we have a randomized lower bound $6e/(6e-1) > 1.06532$ for the online list model problem [15], which is clearly also valid for the semionline problem with known largest size, to the best of the authors' knowledge, randomized algorithm is still unknown. Hence it is promising to devise randomized online and semionline algorithms for the discussed problems. It will also be interesting to devise improved algorithms for other semionline problems of the list model.

REFERENCES

[1] S. Albers, *Better bounds for on-line scheduling*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 130–139.

[2] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra, *New algorithms for an ancient scheduling problem*, J. Comput. System Sci., 51 (1995), pp. 359–366.

[3] Y. Bartal, H. Karloff, and Y. Rabani, *A better lower bound for on-line scheduling*, Inform. Process. Lett., 50 (1994), pp. 113–116.

[4] S. Y. Cai and Y. He, *Quasi-online algorithms for scheduling non-increasing processing-time jobs with processor cost*, Acta Automat. Sinica, 29 (2003), pp. 917–921 (in Chinese).

[5] B. Chen, A. van Vliet, and G. Woeginger, *New lower and upper bounds for on-line scheduling*, Oper. Res. Lett., 16 (1994), pp. 221–230.

[6] U. Faigle, W. Kern, and G. Turan, *On the performance of on-line algorithm for partition problems*, Acta Cybernet., 9 (1989), pp. 107–119.

[7] G. Galambos and G. J. Woeginger, *An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling*, SIAM J. Comput., 22 (1993), pp. 349–355.

[8] R. L. Graham, *Bounds for certain muliprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1582.

[9] Y. He and S. Y. Cai, *Semi-online scheduling with machine cost*, J. Comput. Sci. Tech., 17 (2002), pp. 781–787.

[10] Y. He and G. Zhang, *Semi-online scheduling on two identical machines*, Computing, 62 (1999), pp. 179–187.

[11] C. Imreh and J. Noga, *Scheduling with machine cost*, in Proceedings of RANDOM-APPROX'99, Lecture Notes in Comput. Sci. 1671, Springer-Verlag, New York, 1999, pp. 168–176.

[12] D. R. Karger, S. J. Phillips, and E. Torng, *A better algorithm for an ancient scheduling problem*, J. Algorithms, 20 (1996), pp. 400–430.

[13] H. Kellerer, V. Kotov, M. R. Speranza, and Z. Tuza, *Semi on-line algorithms for the partition problem*, Oper. Res. Lett., 21 (1997), pp. 235–242.

[14] J. F. Rudin III and R. Chandrasekaran, *Improved bounds for the online scheduling problem*, SIAM J. Comput., 32 (2003), pp. 717–735.

[15] S. Seiden, *A guessing game and randomized online algorithms*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 592–601.

[16] S. Seiden, J. Sgall, and G. Woeginger, *Semi-online scheduling with decreasing job sizes*, Oper. Res. Lett., 27 (2000), pp. 215–221.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1052–1066

SOLVING EQUATIONS IN THE RELATIONAL ALGEBRA*

JOACHIM BISKUP†, JAN PAREDAENS‡,
THOMAS SCHWENTICK§, AND JAN VAN DEN BUSSCHE¶

Abstract. Enumerating all solutions of a relational algebra equation is a natural and powerful operation which, when added as a query language primitive to the nested relational algebra, yields a query language for nested relational databases, equivalent to the well-known powerset algebra. We study *sparse* equations, which are equations with at most polynomially many solutions. We look at their complexity and compare their expressive power with that of similar notions in the powerset algebra.

Key words. relational algebra, equation, nested relation, Fagin's theorem, sparse expression, parity

AMS subject classifications. Primary, 68P15; Secondary, 03C13, 68Q17, 05C25

DOI. 10.1137/S0097539701390859

1. Introduction.

Suppose we are allowed to see only a view on a database B, computed by a relational algebra expression e. If we still want to find out what B is, we might try to "invert" e (assuming we know this expression), which will only work when we also know the finite domain D of B. Specifically, we can enumerate all databases X over D, and test for each X whether it satisfies the equation $e(X) = e(B)$. One of these solutions will be B, of course, so if the set of all solutions is not too big, it might provide us with useful information for starting our detective work.

The above simple scenario from database security led us to wonder what can be said in general about the solution of equations in the relational algebra. Generally, if e_1 and e_2 are two algebra expressions over some database schema augmented with some relation variables $X_1, \ldots, X_p$, we can consider the equation $e_1 = e_2$. A *solution* of this equation, given a database B with finite domain D, is a tuple $(X_1, \ldots, X_p)$ of relations over D such that e_1 and e_2 evaluate to the same relation on the augmented database $(B, X_1, \ldots, X_p)$.

Asking whether there exists a solution of a relational algebra equation on a database is almost exactly the same thing as asking whether an existential second-order logic sentence is true on that database. Hence, by Fagin's theorem [Fag74, EF95], the problems that can be formulated as finding a solution of some relational algebra equation are nothing but the problems in NP.

However, in the present paper, we start from the observation that the set of all solutions of an equation, being a set of tuples of relations, is a *nested* relation. One can therefore consider the enumeration of all solutions of an equation as a query language primitive, which can be added to the nested relational algebra. We introduce and study this extension of the nested relational algebra, which we call the *equation*

*Received by the editors June 14, 2001; accepted for publication (in revised form) November 26, 2003; published electronically June 25, 2004.
http://www.siam.org/journals/sicomp/33-5/39085.html
†Computer Science Department, University of Dortmund, D-44227 Dortmund, Germany (biskup@ls6.cs.uni-dortmund.de).
‡Department of Math and Computer Science, University of Antwerp, Middelheimlaan 1, B-2020 Antwerpen, Belgium (jan.paredaens@ua.ac.be).
§Phillipps-Universitat Marburg, Fachbereich Mathematik und Informatik, 35032 Marburg, Germany (tick@informatik.uni-marburg.de).
¶Limburgs Universitair Centrum, B-3590 Diepenbeek, Belgium (jan.vandenbussche@luc.ac.be).

algebra. The equation algebra is extremely powerful: it is equivalent to the well-known powerset algebra for nested relations. Our particular interest, however, is in what can be expressed in the equation algebra by using only equations that have a solution set of polynomial size on each database. We call such equations *sparse*.

Our interest in sparse equations does not stem from time efficiency considerations. Indeed, it is not obvious how knowing that an equation is sparse would help in actually finding even one solution more quickly. Neither is it obvious, however, that it would *not* help. For example, consider the problem of checking on a given database whether some fixed sparse relational algebra equation has a solution. Using an extension of Fagin's theorem to nested relational databases, we show that this problem can be NP-hard only if every problem in NP can already be decided by a polynomial-time nondeterministic Turing machine that has only polynomially many accepting computations on each input. The latter is one of the many unresolved questions in computational complexity theory [All86].

Nevertheless, sparse equations are still interesting from a space efficiency standpoint. Indeed, for the natural evaluation strategy for equation algebra expressions to run in polynomial space, it is necessary that all equations occurring in the expression be sparse. Interest in fragments of powerful query languages for which the natural evaluation strategy is polynomial-space is not new to database theory research. For example, Abiteboul and Vianu [AV91] showed that the parity query is not expressible in the polynomial-space fragment of various computationally complete query languages.

Closer to our topic is the work of Suciu and Paredaens [SP97], who showed that queries such as transitive closure and parity are not expressible in the polynomial-space fragment of the powerset algebra for nested relations. This fragment consists of all powerset algebra expressions where all intermediate results are of polynomial size, on each database. Note that this fragment does make sense, as there are expressions that always produce a result of logarithmic size; applying the (exponential) powerset operator to such expressions produces a result of polynomial size.

We also mention Grumbach and Vianu [GV95], who also studied a sparsity notion in connection with queries over nested relational databases, although they considered sparsity as a property of databases rather than of query language expressions.

Suciu and Paredaens conjectured in general that the polynomial-space fragment of the powerset algebra has no more power than the nested relational algebra without powerset. (This conjecture has been confirmed for monadic database schemas [VdB].) At first sight, the operator that we add to the nested relational algebra, to enumerate all solutions of an equation, does not seem to be that different from the powerset operator. After all, both operators perform some kind of potentially exponential enumeration.

Yet, as we will point out, the analogue of the Suciu–Paredaens conjecture does not hold for the sparse fragment of the equation algebra. Specifically, using sparse equations only, we can express transitive closure; in fact, we can express any fixpoint query. This complements a result by Abiteboul and Hillebrand [AH95], who showed that transitive closure becomes expressible in the powerset algebra in polynomial space, provided we use a more clever "pipelined" evaluation strategy. Actually, every fixpoint query is already expressible using equations that are not just sparse, but even unambiguous: they have a unique solution on each database. Unambiguous equations in the relational algebra are known as *implicit definitions* in first-order logic, and were studied in the context of finite model theory by Kolaitis [Kol90]. Kolaitis already showed that every fixpoint query can be implicitly defined. We offer

a straightforward, direct proof.

Another example of the differences between the sparse fragment of the equation algebra and that of the powerset algebra is given by the well-known nesting operator of the nested relational algebra. This operator becomes redundant once we extend the algebra with the solution operator or with the powerset operator. However, the original nesting operator never blows up exponentially. We show that, without using the nesting operator itself, nesting is expressible in the equation algebra using sparse equations only, but that the same is not possible with a polynomial-space powerset algebra expression.

However, there are also similarities between the two fragments. Specifically, we prove an analogue to the Suciu–Paredaens result, to the effect that the parity query is not expressible in the sparse fragment of the equation algebra either. This is our main technical contribution; the proof is in the style of an elegant argument of Liebeck [Lie83], invoking Bochert's theorem on the order of primitive permutation groups. Coming back to connection with implicit definitions in first-order logic, our result generalizes the known and easy fact that the parity query cannot be defined implicitly [Kol90], in two directions: from unambiguous to sparse, and from a single equation to an arbitrarily complex expression involving several, possibly nested, equations.

This paper is organized as follows. Section 2 recalls the nested relational data model. Section 3 introduces relational algebra equations. Section 4 introduces the equation algebra. Section 5 introduces sparse equations, as well as the natural evaluation strategy for equation algebra expressions. Section 6 studies the time complexity of sparse equations. Finally, section 7 presents the comparison with the polynomial-space powerset algebra.

2. Preliminaries. We quickly recall the nested relational data model and algebra [TF86, AHV95].

Relation types are defined as follows. The symbol 0 is a type, and, if $\tau_1, \ldots, \tau_k$ are types, then so is $(\tau_1, \ldots, \tau_k)$. For a type τ and some set D of atomic values, the *relations of type τ on D* are inductively defined as follows. A relation of type 0 on D is just an element of D (this serves merely as the base case for the induction). A relation of type $(\tau_1, \ldots, \tau_k)$ on D is a set of k-tuples $(x_1, \ldots, x_k)$ such that x_i is a relation of type τ_i on D for $i = 1, \ldots, k$.

A *database schema* is a finite set $\mathcal{S}$ of relation names, where each relation name has an associated type different from 0. A *database* B over $\mathcal{S}$ consists of a nonempty finite domain D of atomic values, together with, for each relation name R in $\mathcal{S}$, a relation R^B of type τ on D, where τ is the type of R.

The operators of the nested relational algebra are those of the standard relational algebra (union $\cup$ and difference $-$ of relations of the same type; Cartesian product $\times$; projection π; selection σ for equality, which can now be set-equality of nested relations), plus the operators *nesting* ν and *unnesting* μ, defined as follows.

Let R be a relation of type $(\tau_1, \ldots, \tau_k)$, and let $i_1, \ldots, i_p \in \{1, \ldots, k\}$. Then the nesting $\nu_{i_1,\ldots,i_p}(R)$ equals the relation

$$\Big\{\Big(x_1, \ldots, x_k, \big\{(y_{i_1}, \ldots, y_{i_p}) \mid (y_1, \ldots, y_k) \in R \text{ and } x_j = y_j \text{ for each } j \in \{1, \ldots, k\} - \{i_1, \ldots, i_p\}\big\}\Big) \;\Big|\; (x_1, \ldots, x_k) \in R\Big\}$$

of type $(\tau_1, \ldots, \tau_k, (\tau_{i_1}, \ldots, \tau_{i_p}))$.

Let R be as in the previous paragraph, and let $i \in \{1, \ldots, k\}$ such that $\tau_i \neq 0$; thus τ_i is of the form $(\omega_1, \ldots, \omega_\ell)$. Then the unnesting $\mu_i(R)$ equals the relation

$$\{(x_1, \ldots, x_k, y_1, \ldots, y_\ell) \mid (x_1, \ldots, x_k) \in R \text{ and } (y_1, \ldots, y_\ell) \in x_i\}$$

of type $(\tau_1, \ldots, \tau_k, \omega_1, \ldots, \omega_\ell)$.

The expressions of the *nested relational algebra* over a schema $\mathcal{S}$ are now built up using the above operators from the relation names in $\mathcal{S}$ and the symbol D, which stands for the finite domain of the input database. The relation to which an expression e evaluates on a database B is denoted by $e(B)$.

One can extend the nested relational algebra to the *powerset algebra* by adding the powerset operator, defined as follows. Let R be a relation of type $(\tau_1, \ldots, \tau_k)$. Then the powerset $\Pi(R)$ equals the relation $\{S \mid S \subseteq R\}$ of type $((\tau_1, \ldots, \tau_k))$.

3. Equations. Let $\mathcal{S}$ and $\mathcal{X}$ be disjoint database schemas; $\mathcal{S}$ is the actual database schema, while $\mathcal{X}$ is thought of as a set of additional relation variables. Let e_1 and e_2 be two expressions over the expanded schema $\mathcal{S} \cup \mathcal{X}$.

DEFINITION 3.1. *Given a database B over $\mathcal{S}$, a* solution to the equation $e_1 = e_2$ *is a database A over $\mathcal{X}$ with the same finite domain as B, such that $e_1(B, A) = e_2(B, A)$.*

Here, (B, A) denotes the expansion of B with A, i.e., the database over $\mathcal{S} \cup \mathcal{X}$ that has the same finite domain as B, that equals B on $\mathcal{S}$, and that equals A on $\mathcal{X}$.

Example 3.2. For a very simple example, let $R \in \mathcal{S}$ and let $\mathcal{X} = \{X\}$, where X has the same type as R. Then $X \cup R = R$ is an equation. Given a database B over $\mathcal{S}$, a database A over $\{X\}$ is a solution if and only if $X^A \subseteq R^B$.

For another example, let X be a relation variable of type $(0, 0)$. One can write a relational algebra expression e such that on any database A over $\{X\}$ with finite domain D, $e(A)$ is empty if and only if X^A is one-to-one, the projections $\pi_1(X^A)$ and $\pi_2(X^A)$ are disjoint, and their union equals D. An example of an e that works is

$$\begin{aligned}\pi_1\sigma_{2\neq 4}\sigma_{1=3}(X \times X) \cup \pi_2\sigma_{2=4}\sigma_{1\neq 3}(X \times X) \\ \cup \big(\pi_1(X) - (\pi_1(X) - \pi_2(X))\big) \\ \cup \big(D - (\pi_1(X) \cup \pi_2(X))\big) \cup \big((\pi_1(X) \cup \pi_2(X)) - D\big).\end{aligned}$$

Then the equation $e = \varnothing$ has a solution on a database B with finite domain D if and only if the cardinality of D is even. (Technically, $e = \varnothing$ is not an equation because the symbol $\varnothing$ is not an expression, but we can easily take $\varnothing$ here to stand for the expression $D - D$, which always evaluates to the empty relation.) □

Remark 3.3. In the above example, we used an equation of the special form $e = \varnothing$. Actually, this form is not so special at all, because any equation $e_1 = e_2$ can be brought in this form as $e_1 \,\Delta\, e_2 = \varnothing$, where $e_1 \,\Delta\, e_2$ stands for $(e_1 - e_2) \cup (e_2 - e_1)$ (symmetric difference).

Alternatively, one might wonder about the use of *dis*equations, of the form $e \neq \varnothing$. These are nothing but equations in disguise, because they can also be written as $\pi_1(D \times e) = D$. Conversely, any equation $e_1 = e_2$ can also be written as the disequation $D - \pi_1(D \times (e_1 \,\Delta\, e_2)) \neq \varnothing$. □

4. The equation algebra. We are now ready to extend the nested relational algebra with a solution operator for equations. We refer to the resulting algebra as the *equation algebra*.

To allow for an elegant definition, we do not fix a schema $\mathcal{S}$ in advance. Rather, we assume a sufficiently large supply of relation names of all possible types. Any relation name can now occur in an expression. Like in logic formulas, some will occur *free* and others will occur *bound*. Bound relation names are bound by the solution of an equation, and serve as the variables of the equation. Within the equation, however, they are still free. We denote the set of relation names that occur free in an equation algebra expression e by $\mathit{free}(e)$.

For the constructs of the nested relational algebra, this is all straightforward: for a relation name R, we have $\mathit{free}(R) := \{R\}$; for expressions e of the form $(e_1 \cup e_2)$, $(e_1 - e_2)$, or $(e_1 \times e_2)$, we have $\mathit{free}(e) := \mathit{free}(e_1) \cup \mathit{free}(e_2)$; for expressions e of the form $\sigma(e')$, $\pi(e')$, $\nu(e')$, or $\mu(e')$, we have $\mathit{free}(e) := \mathit{free}(e')$. For the expression D, we have $\mathit{free}(D) := \varnothing$.

The definition of the new solution operator is now the following.

DEFINITION 4.1. *Let e_1 and e_2 be expressions, and let $X_1, \ldots, X_p$ be a sequence of distinct relation names. Then*

$$\{(X_1, \ldots, X_p) \mid e_1 = e_2\}$$

is also an expression (called a solution expression). We define its free *set as* $(\mathit{free}(e_1) \cup \mathit{free}(e_2)) - \{X_1, \ldots, X_p\}$. *We say that the X_i* become bound.

Note that this is a recursive definition, in the sense that e_1 and e_2 can contain solution operators in turn. To avoid clutter, we disallow equation algebra expressions in which a free relation name at the same time becomes bound in some subexpression, as in $X \times \{(X) \mid X \cup R = R\}$.

An expression e in the equation algebra can be evaluated on databases B over any schema that contains $\mathit{free}(e)$. We already know how this evaluation is defined for the constructs of the nested relational algebra. So we only have to give the following definition.

DEFINITION 4.2. *For a solution expression, e, of the form $\{(X_1, \ldots, X_p) \mid e_1 = e_2\}$, and a database B, the evaluation $e(B)$ equals the relation*

$$\{(X_1^A, \ldots, X_p^A) \mid A \text{ is a database over } \{X_1, \ldots, X_p\} \text{ that is a solution of } e_1 = e_2, \text{ given } B\}.$$

This relation is of type $(\tau_1, \ldots, \tau_p)$, where τ_i is the type of X_i for $i = 1, \ldots, p$.

Example 4.3. Recall the simple example equation $X \cup R = R$ from Example 3.2. We can turn this equation into the following equation algebra expression e: $\{(X) \mid X \cup R = R\}$, or, more readibly, $\{(X) \mid X \subseteq R\}$. Note that $\mathit{free}(e) = \{R\}$. On any database B over $\{R\}$, the relation $e(B)$ equals $\Pi(R^B)$ (recall the powerset operator Π from section 2). In other words, the equation algebra expression e is equivalent to the powerset algebra expression $\Pi(R)$.

The equation algebra allows equations to be used inside equations. For example, if we want to compute the powerset of the powerset of R, we can write:

$$\big\{(Y) \mid Y \subseteq \{(X) \mid X \subseteq R\}\big\}.$$

As a third example, let R and T be relation names of the same binary type (τ, τ) for some τ. One can write a relational algebra expression e_{tc} such that on any database C over $\{R, T\}$, e_{tc} is empty if and only if $R^C \subseteq T^C$ and T^C is transitively closed. One can also write a nested relational algebra expression $e_{\min}$ that selects, out

of a set of binary relations, the minimal ones w.r.t. set inclusion. Explicit forms for e_{tc} and $e_{\min}$ have been given by Gyssens and Van Gucht [GVG]. Then the following equation algebra expression computes the transitive closure of relation R:

$$\pi_{2,3}\mu_1 e_{\min}\big(\{(T) \mid e_{\mathrm{tc}} = \varnothing\}\big).$$

Indeed, the subexpression $\{(T) \mid e_{\mathrm{tc}} = \varnothing\}$ returns the collection of all transitively closed relations on the same domain as R and containing R; applying $e_{\min}$ to that collection results in the singleton consisting of the minimal element, i.e., the transitive closure of R (by definition of transitive closure); applying unnesting μ_1 produces the actual tuples in the transitive closure, keeping the nested relation (cf. our definition of the effect of μ in section 2); and applying $\pi_{2,3}$ finally removes the nested relation. □

In the above example we saw that the powerset operator is expressible in the equation algebra. Conversely, the solution operator is easily expressed in the powerset algebra. Hence, we obtain the following.

PROPOSITION 4.4. *The equation algebra is equivalent to the powerset algebra.*

Proof. To see that $\{(X_1, \ldots, X_p) \mid e_1 = e_2\}$ can be expressed in the powerset algebra, we begin by noting that for any relation type τ one can write a powerset algebra expression Π^τ yielding the collection of all relations of type τ on D. For example, $\Pi^{(0,0)}$ is $\Pi(D \times D)$, and $\Pi^{(0,(0))}$ is $\Pi(D \times \Pi(D))$. Hence, if the type of X_i is τ_i for $i = 1, \ldots, p$, then $\Pi^{\tau_1} \times \cdots \times \Pi^{\tau_p}$ yields the collection of all potential solutions. Now it suffices to observe that one can write nested relational algebra expressions that apply e_1 or e_2 to each database in this collection separately. Explicit forms of such expressions have been given by Gyssens and Van Gucht [GVG]. After that, the actual solutions can be selected by an equality selection. □

5. Sparse equations. So far, the equation algebra is merely another syntax for the powerset algebra, or, if you want, higher-order logic. However, when we consider a natural evaluation strategy for equation algebra expressions, we start to notice some differences.

By the natural strategy to evaluate a solution expression of the form $\{(X_1, \ldots, X_p) \mid e_1 = e_2\}$, we mean the following. Enumerate all databases A over $\{X_1, \ldots, X_p\}$, on the finite domain of the given input database, *one by one, reusing the same space.* For each A we test whether it is a solution (by recursively evaluating e_1 and e_2), and if so, we include it in the result.

For the constructs of the nested relational algebra, the natural evaluation strategy is clear: if we have to evaluate an expression of the form $e_1 \,\Theta\, e_2$, with $\Theta \in \{\cup, -, \times\}$, we create two intermediate results by recursively evaluating e_1 and e_2, and then apply Θ to these two intermediate results. Similarly, if we have to evaluate an expression of the form $\theta(e)$, with $\theta \in \{\pi, \sigma, \nu, \mu\}$ (and parameters added in subscript), we create an intermediate result by recursively evaluating e, and then apply θ to this intermediate result.

In view of this natural evaluation strategy, we now propose the following.

DEFINITION 5.1. *An equation is called* sparse *if all its relation variables are of* flat *type, i.e., of the form* $(0, \ldots, 0)$*, and the number of solutions on any given database is at most polynomial in the size of that database.*

Example 5.2. The two equations from Example 3.2 are not sparse. Probably the simplest example of a nontrivial sparse equation is the following. Let X be a relation name of type (0). One can write a relational algebra expression e over $\{X\}$ such that on any database A over $\{X\}$, $e(A)$ is empty if and only if X^A is a singleton. Then the equation $e = \varnothing$, where X is taken as the relation variable to be solved for, is sparse.

Indeed, given any database B with finite domain D, the solutions are precisely all singleton subsets of D. There clearly are only a linear (and thus at most polynomial) number of possible solutions. □

Remark 5.3. A natural alternative definition of sparsity would be the one where, in Definition 5.1, we would look only at databases over the schema consisting of the relation names that actually occur free in the equation. One easily sees, however, that this alternative definition yields the same notion of sparsity. □

Sparse equations are connected to the natural evaluation strategy in the following way.

PROPOSITION 5.4. *The natural strategy for evaluating an equation algebra expression e runs in polynomial space if and only if all equations occurring in e are sparse.*

Here, we count not only the space occupied by the intermediate results stored during evaluation, but also the size of the final result.

Proof. The if direction is clear. For the only-if direction, we work by induction on the nesting depth of equations. The base case—expressions that do not contain any equations at all—is trivial.

For the inductive step, consider a top-level equation $\{(X_1, \ldots, X_p) \mid e_1 = e_2\}$ occurring in e. The natural strategy for evaluating this equation runs in polynomial space, so in particular, for each expansion of each database B over $\mathit{free}(e)$ with a candidate solution A over $\{X_1, \ldots, X_p\}$, the natural evaluation of e_1 and e_2 on (B, A) runs in polynomial space. In this way we consider every possible database C over $\mathit{free}(e) \cup \{X_1, \ldots, X_p\}$, because the restriction of C to $\mathit{free}(e)$ is a possible B, and C itself then is a possible expansion of B. Hence, the natural evaluation strategies of e_1 and e_2 in general run in polynomial space.

Formally, we must note here that e_1 and e_2 might not actually mention certain relation names in $\mathit{free}(e)$ or $\{X_1, \ldots, X_p\}$, and that there is still the formal possibility that their natural evaluation might not run in polynomial space on databases over schemas not containing these names. However, using Remark 5.3, it can be seen that this is impossible.

By induction, we can therefore conclude that all equations occurring nested inside a top-level equation are sparse.

The top-level equation itself must also be sparse. In proof, if one of the X_i would be of nonflat type, even one candidate solution can already be of exponential size. Indeed, even in the simplest case where X_i would be of type $((0))$, on a domain with n elements, a possible candidate value for X_i is the collection of all subsets of that domain, which is of size 2^n. Thus, every X_i is of flat type. Furthermore, since we store the solution set as an intermediate result, it must be of at most polynomial size on all databases over $\mathit{free}(e)$. Since the individual solutions are flat databases and thus of polynomial size, the cardinality of the solution set must therefore be at most polynomial. □

6. Time complexity of equation nonemptiness. The *time* complexity of solving sparse equations is closely linked to an open question from computational complexity theory. Unlike the previous section, in this section we are not talking about the natural evaluation strategy, whose time complexity is clearly at least exponential as soon as there are equations to be solved. Instead, we will be looking at the time complexity of the *nonemptiness problem* of equations.

The nonemptiness problem of an equation over a schema $\mathcal{S}$ with relation variables $\mathcal{X}$ is the problem of deciding, given a database over $\mathcal{S}$, whether the equation has a

solution on that database. *In the present section we will consider only equations that do not contain equations inside.*

Let us begin by considering equations that are not necessarily sparse but that still have only flat variables. The nonemptiness problem of such a flat-variable equation is clearly in NP. Now suppose, moreover, that the database schema $\mathcal{S}$ is also flat; then $\mathcal{S} \cup \mathcal{X}$ (the expansion of $\mathcal{S}$ with the relation variables of the equation) is an entirely flat schema. Of course, the equation $e_1 = e_2$ is still in general in the *nested* relational algebra; i.e., e_1 and e_2 can contain ν and μ operators. A result by Paredaens and Van Gucht [PVG92], however, implies that the nested relational algebra condition $e_1 = e_2$ can also be expressed in the form $e \neq \varnothing$, with e a *flat* relational algebra expression. The nonemptiness problem of the equation thus amounts to asking whether $\{(X_1, \ldots, X_p) \mid e \neq \varnothing\}$ is nonempty on a given database B over $\mathcal{S}$. Equivalently, we ask whether the existential second-order logic ($\exists$SO) sentence $\exists X_1 \ldots \exists X_p \, \varphi_e$ is true on B, where φ_e is a first-order logic sentence expressing that $e \neq \varnothing$. Moreover, by the equivalence of relational algebra and first-order logic, *any* $\exists$SO property can be obtained in this way. Now, Fagin's theorem [Fag74, EF95] states that $\exists$SO captures exactly the NP properties of flat relational databases. Hence, *the class of nonemptiness problems of flat-variable equations over flat database schemas is exactly the class of NP properties of flat relational databases.*

What if $\mathcal{S}$ is not necessarily flat? We next show that we still get exactly NP. In essence, this is an extension of Fagin's theorem to nested relational databases.

PROPOSITION 6.1. *Every property of nested relational databases over some fixed schema $\mathcal{S}$ that is in NP and closed under isomorphism corresponds to the nonemptiness problem of some flat-variable equation over $\mathcal{S}$.*

Proof. The crux is a representation of nested relational databases by "pseudoflat" ones, also used by Gyssens, Suciu, and Van Gucht [GSVG01]. Given a nested relational database B, we define its *extended domain*, denoted by $\mathrm{edom}(B)$, as the union of its finite domain of atomic values with the set of all relations occurring (possibly deeply nested) in B. We regard the relations in the extended domain as if they were atomic values. Now for any nested relational database schema $\mathcal{S}$ we can construct a flat one $\bar{\mathcal{S}}$, together with a mapping *rep* from the set of databases over $\bar{\mathcal{S}}$ *onto* the set of databases over $\mathcal{S}$, expressible in the nested relational algebra. The details of this mapping need not concern us here. Important is that we can furthermore construct a converse mapping *flat* from the set of databases over $\mathcal{S}$ to the set of databases over $\bar{\mathcal{S}}$, also expressible in the nested relational algebra, with the following properties for each database B over $\mathcal{S}$: (1) the finite domain of $\mathit{flat}(B)$ equals $\mathrm{edom}(B)$; and (2) $\mathit{rep}(\mathit{flat}(B)) = B$. Note that while *flat* is expressed in the nested relational algebra, the result $\mathit{flat}(B)$ is not really a flat database, because of the nested relations in the extended domain. However, it is "pseudoflat," in the sense that we regard these relations as if they were atomic values. For any relation name R of $\bar{\mathcal{S}}$, we denote the nested relational algebra expression defining the R-component of the mapping *flat* by flat_R. Likewise, we denote the expression defining the D-component by flat_D.

Given this representation, the proof is straightforward. Let L be an NP property of databases over $\mathcal{S}$, closed under isomorphism. Define the property $\bar{L}$ of databases over $\bar{\mathcal{S}}$ as follows: F satisfies $\bar{L}$ if $\mathit{rep}(F)$ satisfies L. Then $\bar{L}$ is in NP and is also closed under isomorphism. Hence, Fagin's theorem gives us an $\exists$SO sentence $\exists X_1 \ldots \exists X_p \, \varphi$ over $\bar{\mathcal{S}}$ expressing $\bar{L}$. By the equivalence of relational algebra and first-order logic, there is a flat relational algebra expression e over $\bar{\mathcal{S}} \cup \{X_1, \ldots, X_p\}$ such that the first-order logic sentence φ is equivalent to $e \neq \varnothing$. Now modify e as follows: for every relation name R of $\bar{\mathcal{S}}$, replace every occurrence of R in e by flat_R. Likewise, replace

every occurrence of D in e by flat_D. Denote the resulting nested relational algebra expression by e'.

We now have, for any database B over $\mathcal{S}$, that B satisfies L if and only if $\exists X_1 \dots X_p\, e' \neq \varnothing$ is true on B. The condition $e' \neq \varnothing$ can easily be written as an equation (cf. Remark 3.3). □

We are now ready to turn to sparse equations. Their nonemptiness problem is not just in NP, but actually in the complexity class *FewP* [All86], consisting of all problems that can be decided by a polynomial-time nondeterministic Turing machine that has at most polynomially many accepting computations on each input. Clearly, $\mathrm{P} \subseteq \mathrm{FewP} \subseteq \mathrm{NP}$, but the strictness of these inclusions remains open.

The obvious question to ask is whether Proposition 6.1 remains true if we focus on sparse equations, and replace "NP" by "FewP." The answer is an easy "yes," but then we must restrict our attention to *ordered* databases: databases that include a total order on their finite domain as one of their relations.

PROPOSITION 6.2. *Every property of ordered nested relational databases over some fixed schema $\mathcal{S}$ that is in FewP and closed under isomorphism corresponds to the nonemptiness problem of some sparse equation over $\mathcal{S}$, when restricted to ordered databases only.*

Proof. The usual proof of Fagin's theorem immediately yields the case where $\mathcal{S}$ is flat. Indeed, in that proof, to express an NP property decided by some polynomial-time bounded nondeterministic Turing machine M, one writes an $\exists$SO sentence $\exists X_1 \exists X_2 \dots \exists X_p\, \varphi$, where X_1 stands for an order on the domain, $X_2, \dots, X_p$ encode (using the order in X_1) a computation of M, and φ checks whether the computation is accepting. As we are dealing with a FewP property, M has only polynomially many accepting computations. Hence, the equation $\{(X_1, \dots, X_p) \mid \varphi\}$ would be sparse, were it not for X_1, as there are exponentially many possible orders on a finite domain. On ordered databases, however, there is no need for X_1, and we obtain a genuinely sparse equation.

This is for flat databases; for general nested relational databases we use the same representation technique as in the proof of Proposition 6.1. □

As a corollary we get the following.

COROLLARY 6.3. *There exists a sparse equation whose nonemptiness problem is NP-complete, if and only if* $\mathrm{FewP} = \mathrm{NP}$.

7. Sparse equations versus sparse powerset expressions. Naturally, we call an equation algebra expression sparse if all equations occurring in it are sparse. Inspired by Proposition 5.4, we can also define a sparsity condition on powerset algebra expressions: call a powerset algebra expression *sparse* if its natural evaluation strategy (defined in the obvious way) runs in polynomial space.

Remark 7.1. Using standard techniques, one can show that sparsity is undecidable for equation algebra expressions as well as powerset algebra expressions. An interesting question, raised by an anonymous referee, is whether one can give useful syntactic restrictions that guarantee sparsity. Ideally every sparse expression would be equivalent to one satisfying the syntactic restrictions. □

Suciu and Paredaens [SP97] showed that transitive closure of a flat binary relation is not expressible by a sparse powerset expression. In Example 4.3, we gave an obvious equation algebra expression for transitive closure, but that expression was not sparse. We can do better, as in the following.

PROPOSITION 7.2. *Transitive closure of a flat relation is expressible by a sparse equation algebra expression.*

Proof. Given a binary relation R and a natural number $n \geqslant 1$, we define the relation R^n as $R \circ \cdots \circ R$ (n times R), where $\circ$ is the classical composition operator of binary relations: $S \circ T = \pi_{1,4}\sigma_{2=3}(S \times T)$. Further, define $R^{\leqslant n}$ as $\bigcup_{i=1}^{n} R^i$, and define $R^{=n}$ as $R^n - R^{\leqslant n-1}$. Note that $R^{\leqslant |R|}$ equals the transitive closure of R, and that for $n > |R|$, $R^{\leqslant n} = R^{\leqslant |R|}$.

Now consider the following 6-ary relation *Run*:

$$Run := \bigcup_{i=1}^{|R|} R^{\leqslant i} \times R^{\leqslant i+1} \times R^{=i+1}.$$

We show next that there is an equation whose *only* solution, given R, is *Run*. This proves the proposition, because all we then have to do is unnest the solution set and project on the middle two columns to get the transitive closure. (The only exception is when *Run* is empty, in which case the transitive closure of R is R itself, but this can also easily be tested in the nested relational algebra.)

The desired equation expresses the conjunction of the following conditions on relation variable X:

1. For any pair $(x_5, x_6) \in \pi_{5,6}(X)$, we denote the relation

$$\{(x_1, x_2, x_3, x_4) \mid (x_1, \ldots, x_6) \in X\}$$

by $\tilde{X}(x_5, x_6)$, and denote further

$$\hat{X}(x_5, x_6) := \pi_{1,2}(\tilde{X}(x_5, x_6)) \quad \text{and}$$
$$\check{X}(x_5, x_6) := \pi_{3,4}(\tilde{X}(x_5, x_6)).$$

Then for every $(x, y) \in \pi_{5,6}(X)$ we must have
   (a) $\tilde{X}(x, y) = \hat{X}(x, y) \times \check{X}(x, y)$;
   (b) $\hat{X}(x, y) \supseteq R$;
   (c) $\check{X}(x, y) = \hat{X}(x, y) \cup \hat{X}(x, y) \circ R$; and
   (d) $(x, y) \in \check{X}(x, y) - \hat{X}(x, y)$.
   (e) Furthermore, *every* pair (x', y') in difference (d) above belongs to $\pi_{5,6}(X)$, with $\hat{X}(x', y') = \hat{X}(x, y)$ (and thus also $\check{X}(x', y') = \check{X}(x, y)$).
2. $R^{=2} \subseteq \pi_{5,6}(X)$, and for every $(x, y) \in R^{=2}$ we have $\hat{X}(x, y) = R$.
3. For every $(x, y) \in \pi_{5,6}(X)$ such that $\check{X}(x, y) \circ R - \check{X}(x, y) \neq \varnothing$, there exists a pair $(x', y') \in \pi_{5,6}(X)$ with $\hat{X}(x', y') = \check{X}(x, y)$.
4. For every $(x, y) \in \pi_{5,6}(X)$ such that $\hat{X}(x, y) \neq R$, there exists a pair $(x', y') \in \pi_{5,6}(X)$ with $\check{X}(x', y') = \hat{X}(x, y)$.

The conjunction of the above conditions expresses that X equals *Run*. Indeed, by (2), (1c), and (1a) we know that $R^{\leqslant 1} \times R^{\leqslant 2} \times R^{=2} \subseteq X$. By induction and by (3), (1c), (1e), and (1a) we know that $R^{\leqslant i} \times R^{\leqslant i+1} \times R^{=i+1} \subseteq X$ and hence $Run \subseteq X$. Moreover, for every $(x, y) \in R^{=i+1}$ we have $\{(x_1, y_1, x_2, y_2) \mid (x_1, y_1, x_2, y_2, x, y) \in X\} = R^{\leqslant i} \times R^{\leqslant i+1}$. On the other hand, if $(x, y) \in \pi_{5,6}(X)$, then $\hat{X}(x, y) = R$, in which case $(x, y) \in R^{=2}$ by (1c) and (1d), or $\hat{X}(x, y) \neq R$, in which case we know by induction and by (4) and (1b) that $(x, y) \in R^{=i+1}$ for some i. This proves $X = Run$. □

Remark 7.3. The equation constructed in the above proof is not only sparse, it is *unambiguous*: it has a unique solution on each input database. Moreover, the same proof works more generally for any *fixpoint query* [AHV95] on flat databases.

The only difficulty is that fixpoint queries start from the empty relation, while in our proof of Proposition 7.2 we start from R, but that is easily dealt with. As already explained in the Introduction, we thus basically rediscovered an earlier result by Kolaitis to the effect that every fixpoint query is implicitly definable in first-order logic [Kol90]. But note the directness of our proof, straightforwardly specifying the run of the fixpoint computation in an unambiguous way. The original proof (also presented by Ebbinghaus and Flum [EF95]) is a bit more roundabout, specifying the "stage comparison" relation instead. □

Another, perhaps a bit frivolous, example of a query that is expressible using sparse equations but not using sparse powerset expressions is the nesting operator ν. It is easy to express ν in the powerset algebra using the powerset operator and the other operators, but not ν itself; so ν is not primitive in the powerset algebra. As a consequence (Proposition 4.4), ν is not primitive in the equation algebra either. We next observe that when we restrict our analysis to sparse expressions, nesting remains imprimitive in the equation algebra, but becomes primitive again in the powerset algebra.

PROPOSITION 7.4. *Nesting is not expressible by a sparse powerset expression without using the ν operator itself.*

Proof. Suppose we want to express nesting of a flat binary relation R. The first application of the powerset operator is to the result of a flat relational algebra expression e applied to R. Let us focus on the case where R is the identity relation on a finite domain of n elements. A straightforward argument by structural induction shows that, on identity relations, every relational algebra expression is equivalent to a finite disjunction of *equality types*. Here, an equality type is a maximally consistent conjunction of equalities $x_i = x_j$ and nonequalities $x_i \neq x_j$ over the variables $x_1, \ldots, x_k$, where k is the output arity of e. We thus see that either $e(R)$ is empty on all such R (this is when the disjunction is empty), or $e(R)$ is of size at least n when n is at least k. In the empty case, the powerset operator is useless, and we continue to the next application of powerset. Otherwise, the powerset operator explodes, and the overall expression is not sparse. □

PROPOSITION 7.5. *Nesting is expressible by a sparse equation algebra expression without using the ν operator itself.*

Proof. Let R be a relation name of type $(0,0)$, and let X and Y be relation variables of type (0). We can write a relational algebra expression e such that on any database C over $\{R, X, Y\}$, $e(C)$ is empty if and only if X is a singleton $\{x\}$ with $x \in \pi_1(R)$, and $Y = \{y \mid (x, y) \in R\}$. An example of an e that works is (Δ stands for symmetric difference)

$$\pi_1\sigma_{1\neq 2}(X \times X) \cup (X - \pi_1(R)) \cup (Y \,\Delta\, \pi_3\sigma_{1=2}(X \times R)).$$

Hence, the expression

$$\mu_1\big(\{(X, Y) \mid e = \varnothing\}\big)$$

is a sparse equation algebra expression equivalent to $\nu_2(R)$.

The construction for general nesting operations is analogous. □

Our final, and main technical, contribution concerns the parity query. Suciu and Paredaens showed that the parity of the cardinality of a finite set is not expressible by a sparse powerset expression. We show the analogue for the equation algebra as follows.

PROPOSITION 7.6. *The parity query is not expressible by a sparse equation algebra expression.*

Proof. Suppose, to the contrary, that we have a sparse equation algebra expression to express the parity of the cardinality of a finite domain D. We may assume that the input schema is empty; i.e., an input database consists of D and nothing else. Consider an innermost equation E_0 occurring in our expression. It may be nested inside other equations $E_1, \ldots, E_k$, enumerated from the inside to the outside. By Remark 3.3, for each $j \in \{0, \ldots, k\}$, E_j can be written in the form $\{(X_1^j, \ldots, X_{i_j}^j) \mid e_j \neq \varnothing\}$, for some i_j, where e_j is a flat relational algebra expression over the flat schema $\{X_1^j, \ldots, X_{i_j}^j\}$ possibly expanded with certain free variables X_m^l, where $l > j$ and $m \leq i_l$.

By our assumption, there are at most polynomially many solutions to equation E_k. For each solution A_k of E_k there are at most polynomially many solutions A_{k-1} of e_{k-1} and so on. A sequence $\bar{A} = (A_k, \ldots, A_0)$ of databases is a *solution vector for* E_0 if A_k is a solution for E_k, and each A_j, $j < k$, is a solution for E_j, given $A_k, \ldots, A_{j+1}$.

Now let $\bar{A} = (A_k, \ldots, A_0)$ be a solution vector for E_0, given an input D of size n. Then for every permutation f of D, $f(\bar{A}) = (f(A_k), \ldots, f(A_0))$ is also a solution vector for E_0. The number of different such $f(\bar{A})$ is precisely $n!/|\mathrm{Aut}(\bar{A})|$, where $\mathrm{Aut}(\bar{A})$ is the group of automorphisms of $\bar{A}$. Since all equations are supposed to be sparse, this number is at most n^ℓ for some fixed ℓ, or, equivalently, $|\mathrm{Aut}(\bar{A})| \geqslant n!/n^\ell$. Setting $k = \ell + 1$, this implies $|\mathrm{Aut}(\bar{A})| \geqslant (n-k)!$ for sufficiently large n.

We thus need to know more about large permutation groups. The following crucial lemma will give us the information we need. The group of permutations of a finite set D is denoted by $\mathrm{Sym}(D)$, and its alternating subgroup of even permutations by $\mathrm{Alt}(D)$. If G is a subgroup of $\mathrm{Sym}(D)$, a *fixed set* for G is a subset $\Delta \subseteq D$ such that every $g \in G$ maps Δ to Δ. The action of G on a fixed set Δ (as a subgroup of $\mathrm{Sym}(\Delta)$) is denoted by G^Δ.

LEMMA 7.7. *Let k be a fixed natural number. Let G be a subgroup of* $\mathrm{Sym}(D)$*, $|D| = n$, n sufficiently large. Then $|G| \geqslant (n-k)!$ implies the existence of a fixed set Δ with $|\Delta| \geqslant n-k$, such that G^Δ contains* $\mathrm{Alt}(\Delta)$.

Proof of Lemma 7.7. For background on finite permutation groups, we refer to Wielandt's book [Wie64], but here are a few preliminaries. An *orbit* of a permutation group G on a set D is a set of the form $\{g(x) \mid g \in G\}$, for some $x \in D$. We call G *transitive* if D is one single orbit of G. Further, G is called *primitive* if it is transitive and has no nontrivial blocks. Here, a *block* of G is a subset $\Delta \subseteq D$ such that for all $g \in G$ the set $g(\Delta)$ is either equal to Δ or disjoint from it. *Trivial* blocks are $\varnothing$, D, and the singletons. If G is not primitive but transitive, there is always a *complete block system* which partitions D into equal-sized nontrivial blocks. We recall the following.

BOCHERT'S THEOREM (1889). *Let G be primitive on D, not containing* $\mathrm{Alt}(D)$*. Let $|D| = n$. Then $|G| \leqslant n!/\lceil n/2 \rceil !$.*

For the proof of the lemma, first assume that G is transitive. There are two possibilities:

1. G is imprimitive with, say, b blocks of size a ($a > 1$, $b > 1$, $ab = n$). Then $|G| \leqslant b!\,(a!)^b$, which in turn is at most $2(\lfloor n/2 \rfloor !)^2$ for n sufficiently large. Thus, by what is given about $|G|$,

   $$(n-k)! \leqslant 2(\lfloor n/2 \rfloor !)^2. \tag{7.1}$$

   However, this is impossible for n sufficiently large.
2. G is primitive. Then, unless G contains $\mathrm{Alt}(D)$ (in which case the lemma is

proved), by Bochert's theorem,

$$(n-k)! \leqslant \frac{n!}{\lceil n/2 \rceil !}. \tag{7.2}$$

Again, this is impossible for n sufficiently large.

Now assume G is intransitive. Let Δ be an orbit of G of maximal size; let $\ell \geqslant 1$ be such that the size of Δ equals $n-\ell$. We have $|G| \leqslant (n-\ell)!\,\ell!$. Suppose $\ell > k$. Then $(n-\ell)!\,\ell!$ reaches its maximum at $\ell = k+1$. Hence, $(n-k)! \leqslant |G| \leqslant (n-k-1)!\,(k+1)!$, and thus $n-k \leqslant (k+1)!$, which is impossible for large enough n.

Thus, $\ell \leqslant k$, or in other words, the size of Δ is at least $n-k$. We have $|G| \leqslant \ell!\,|G^{\Delta}| \leqslant k!\,|G^{\Delta}|$, and so $|G^{\Delta}| \geqslant (n-k)!/k!$. By definition, G^{Δ} is transitive on Δ. We can now apply the same arguments as in the case "G is transitive" above, for G^{Δ} instead of G, and get that G^{Δ} must contain $\mathrm{Alt}(\Delta)$. Indeed, in the right-hand sides of inequalities (7.1) and (7.2), n now becomes $n-\ell$, which has for effect that the upper bounds become smaller. Hence, as these inequalities already were impossible, they now become even more impossible. The extra factor of $1/k!$ in the left-hand sides does not have a significant influence. □

Invoking this lemma for $G = \mathrm{Aut}(\bar{A})$, we get a fixed set Δ of size at least $n-k$ such that any even permutation of Δ can be extended to an automorphism of $\bar{A}$.

Now let X be one of the relation variables of an equation E_j, of arity, say, r, and consider any r-tuple t whose components either are the symbol $*$ or are in $D-\Delta$. Let r' be the number of components that are the symbol $*$. Further, let ξ be an equality type of r'-tuples. Denote by $\mathit{Neighbors}_X^{\bar{A}}(t,\xi)$ the set of r'-tuples over Δ of equality type ξ such that, if we replace the $*$-components of t by the components of the r'-tuple (from left to right), we get a tuple in $X^{\bar{A}}$.

Claim 7.8. *$\mathit{Neighbors}_X^{\bar{A}}(t,\xi)$ either is empty or consists of* all *r'-tuples over Δ of equality type ξ.*

Proof of Claim 7.8. Suppose to the contrary that $N := \mathit{Neighbors}_X^{\bar{A}}(t,\xi)$ is neither empty nor full. Take h_1 in N, and take h_2 (of arity r' and of type ξ) not in N. Take two arbitrary elements from Δ that appear neither in h_1 nor in h_2, and remove them from Δ, resulting in Δ'. Take a third tuple h_3 (of the right arity and type) over Δ', and disjoint from h_1 and h_2. If h_3 is in N, initialize the set I to $\{h_2, h_3\}$; otherwise, set $I := \{h_1, h_3\}$. Now complete I to a maximal set of pairwise disjoint r'-tuples over Δ' of equality type ξ. There are at least $(n-k-2)/r'$ tuples in I.

Assume that at least half of I is outside N; denote the set of these by I'. (The case where at least half of I is *in* N is symmetric.) Fix an $h \in I \cap N$. For each tuple s in I', we consider the permutation ϖ_s that transposes s and h and leaves everything else fixed. If ϖ_s happens to be odd, we make it even by adding the transposition of the two dummy elements we took out of Δ (when we defined Δ'). Then each set $\varpi_s(N)$ contains s but does not contain any other tuples from I'. Thus, we produce in this way at least $f(n) := (n-k-2)/2r'$ different sets of r'-tuples over Δ. Since they are even, each ϖ_s can be extended to an automorphism. Hence, each of the $f(n)$ sets must be the $\mathit{Neighbors}_X^{\bar{A}}(t',\xi)$ of some t'. However, there are less than $(k+1)^r$ different possibilities for t', while $f(n)$ is larger than $(k+1)^r$ for n sufficiently large. Thus we get to the desired contradiction. □

We call (t,ξ) an *r-ary pattern*. If $\mathit{Neighbors}_X^{\bar{A}}(t,\xi)$ is nonempty (and thus full), we say that the pattern is *instantiated in* $X^{\bar{A}}$. Note also that the extreme cases, where t consists exclusively of stars or where t has no star at all, are also allowed and make sense.

By the above, we thus see that any solution vector $\bar{A}$ can be generated by the following nondeterministic procedure:

1. Initialize all relations of $\bar{A}$ to empty.
2. Choose at most k different elements from D, playing the role of the elements outside Δ.
3. For every relation variable X (of arity r, say), run through all r-ary patterns, and for each of them, choose nondeterministically to instantiate it in $X^{\bar{A}}$ or not.

Since k and the number of relation variables are fixed, the number of possible patterns is also fixed. Hence, we can write an expression in the nested relational algebra that, given D, constructs the set of all possible vectors $\bar{A}$ of the above nondeterministic procedure. This set is a superset of the actual set of solution vectors for E_0. Equation E_0 can now be replaced by a nested relational algebra expression which (1) constructs the set of solution candidates $\bar{A}$, (2) projects out the relations for $X_1^0, \ldots, X_{i_0}^0$, and (3) selects those relations which fulfil E_0. The latter is an easy task for the nested relational algebra [GVG].

Hence, we can get rid of E_0. Repeating this process, we can get rid of all equations, so that in the end we are left with a standard nested relational algebra expression for the parity query. But this is well known to be impossible [AHV95, PVG92]. □

Remark 7.9. Note that we have actually shown that over the empty schema, where databases consist of a finite domain and nothing else, the sparse equation algebra is no more powerful than the standard relational algebra. As a matter of fact, the proof can easily be generalized to apply also to schemas having only relation names of type (0). □

Acknowledgment. We are indebted to László Babai, who pointed us to Liebeck's paper.

REFERENCES

[AH95] S. Abiteboul and G. Hillebrand, *Space usage in functional query languages*, in Database Theory—ICDT'95, G. Gottlob and M Y. Vardi, eds., Lecture Notes in Comput. Sci. 893, Springer-Verlag, New York, 1995, pp. 439–454.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, Reading, MA, 1995.

[All86] E. Allender, *The complexity of sparse sets in P*, in Structure in Complexity Theory, A. L. Selman, ed., Lecture Notes in Comput. Sci. 223, Springer-Verlag, New York, 1986, pp. 1–11.

[AV91] S. Abiteboul and V. Vianu, *Generic computation and its complexity*, in Proceedings of the 23rd ACM Symposium on the Theory of Computing, New Orleans, LA, 1991, pp. 209–219.

[EF95] H.-D. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer, New York, 1995.

[Fag74] R. Fagin, *Generalized first-order spectra and polynomial-time recognizable sets*, in Complexity of Computation (Proceedings of the SIAM–AMS Symposium on Applied Math, New York, 1973), R. M. Karp, ed., SIAM-AMS Proceedings 7, AMS, Providence, RI, 1974, pp. 43–73.

[GSVG01] M. Gyssens, D. Suciu, and D. Van Gucht, *Equivalence and normal forms for the restricted and bounded fixpoint in the nested algebra*, Inform. and Comput., 164 (2001), pp. 85–117.

[GV95] S. Grumbach and V. Vianu, *Tractable query languages for complex object databases*, J. Comput. System Sci., 51 (1995), pp. 149–167.

[GVG] M. Gyssens and D. Van Gucht, *The powerset algebra as a natural tool to handle nested database relations*, J. Comput. System Sci., 45 (1992), pp. 76–103.

[Kol90] Ph. G. Kolaitis, *Implicit definability on finite structures and unambiguous computations*, in Proceedings of the 5th IEEE Symposium on Logic in Computer Science, Philadelphia, 1990, pp. 160–180.

[Lie83] M. W. Liebeck, *On graphs whose full automorphism group is an alternating group or a finite classical group*, Proc. London Math. Soc., 47 (1983), pp. 337–362.

[PVG92] J. Paredaens and D. Van Gucht, *Converting nested algebra expressions into flat algebra expressions*, ACM Trans. Database Systems, 17 (1992), pp. 65–93.

[SP97] D. Suciu and J. Paredaens, *The complexity of the evaluation of complex algebra expressions*, J. Comput. System Sci., 55 (1997), pp. 322–343.

[TF86] S. Thomas and P. Fischer, *Nested relational structures*, in The Theory of Databases, P. Kanellakis, ed., JAI Press, Greenwich, CT, 1986, pp. 269–307.

[VdB] J. Van den Bussche, *Simulation of the nested relational algebra by the flat relational algebra, with an application to the complexity of evaluating powerset algebra expressions*, Theoret. Comput. Sci., 254 (2001), pp. 363–377.

[Wie64] H. Wielandt, *Finite Permutation Groups*, Academic Press, New York, 1964.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1067–1092

EQUIVALENCES AND SEPARATIONS BETWEEN QUANTUM AND CLASSICAL LEARNABILITY*

ROCCO A. SERVEDIO† AND STEVEN J. GORTLER‡

Abstract. We consider quantum versions of two well-studied models of learning Boolean functions: Angluin's model of exact learning from membership queries and Valiant's probably approximately correct (PAC) model of learning from random examples. For each of these two learning models we establish a polynomial relationship between the number of quantum or classical queries required for learning. These results contrast known results that show that testing black-box functions for various properties, as opposed to learning, can require exponentially more classical queries than quantum queries. We also show that, under a widely held computational hardness assumption (the intractability of factoring Blum integers), there is a class of Boolean functions which is polynomial-time learnable in the quantum version but not the classical version of each learning model. For the model of exact learning from membership queries, we establish a stronger separation by showing that if any one-way function exists, then there is a class of functions which is polynomial-time learnable in the quantum setting but not in the classical setting. Thus, while quantum and classical learning are equally powerful from an information theory perspective, the models are different when viewed from a computational complexity perspective.

Key words. computational learning theory, quantum computation, PAC learning, query complexity

AMS subject classifications. 68Q32, 68T05, 81P68

DOI. 10.1137/S0097539704412910

1. Introduction.

1.1. Motivation. In recent years many researchers have investigated the power of quantum computers which can query a black-box oracle for an unknown function [1, 5, 6, 9, 14, 10, 11, 15, 17, 20, 21, 24, 40, 46]. The broad goal of research in this area is to understand the relationship between the number of quantum or classical oracle queries which are required to answer various questions about the function computed by the oracle. For example, a well-known result due to Deutsch and Jozsa [17] shows that exponentially fewer queries are required in the quantum model in order to determine with certainty whether a black-box oracle computes a constant Boolean function or a function which is balanced between outputs 0 and 1. More recently, several researchers have studied the number of quantum oracle queries which are required to determine whether the function computed by a black-box oracle is identically zero [5, 6, 9, 15, 24, 46].

A natural question which arises in this framework is the following: what is the relationship between the number of quantum and classical oracle queries which are

*Received by the editors February 1, 2004; accepted for publication (in revised form) February 23, 2004; published electronically June 25, 2004. This paper includes results originally presented in [37] and [38].

http://www.siam.org/journals/sicomp/33-5/41291.html

†Department of Computer Science, Columbia University, New York, NY 10027 (rocco@cs.columbia.edu). This author's research was supported in part by an NSF Research Fellowship, by NSF grant CCR-98-77049, and by the National Security Agency (NSA) and Advanced Research and Development Activity (ARDA) under Army Research Office (ARO) contract DAAD19-01-1-0506. This work was performed while this author was at the Division of Engineering and Applied Sciences of Harvard University.

‡Division of Engineering and Applied Sciences, Harvard University, 33 Oxford Street, Cambridge, MA 02138 (sjg@cs.harvard.edu).

required in order to *exactly identify* the function computed by a black-box oracle? Here the goal is not to determine whether a black-box function satisfies some particular property, such as ever taking a nonzero value, but rather to precisely identify an unknown black-box function from some restricted class of possible functions. The classical version of this problem has been well studied in the computational learning theory literature [2, 12, 22, 26, 27] and is known as the problem of *exact learning from membership queries*. The question stated above can thus be rephrased as follows: what is the relationship between the number of quantum or classical membership queries which are required for exact learning? We answer this question in this paper.

In addition to the model of exact learning from membership queries, we also consider a quantum version of Valiant's widely studied PAC (probably approximately correct) learning model, which was introduced by Bshouty and Jackson [13]. While a learning algorithm in the classical PAC model has access to labeled examples drawn from some fixed probability distribution, a learning algorithm in the quantum PAC model has access to some fixed quantum superposition of labeled examples. Bshouty and Jackson gave a polynomial-time algorithm for a particular learning problem in the quantum PAC model, but did not address the general relationship between the number of quantum versus classical examples which are required for PAC learning. We answer this question as well.

1.2. Our results. We show that in an information-theoretic sense, quantum and classical learning are equivalent up to polynomial factors: for both the model of exact learning from membership queries and the PAC model, there is no learning problem which can be solved using significantly fewer quantum queries than classical queries. More precisely, our first main theorem is the following.

THEOREM 1. *Let $\mathcal{C}$ be any class of Boolean functions over $\{0,1\}^n$, and let D and Q be such that $\mathcal{C}$ is exact learnable from D classical membership queries or from Q quantum membership queries. Then $D = O(nQ^3)$.*

Our second main theorem is an analogous result for quantum versus classical PAC learnability.

THEOREM 2. *Let $\mathcal{C}$ be any class of Boolean functions over $\{0,1\}^n$, and let D and Q be such that $\mathcal{C}$ is PAC learnable from D classical examples or from Q quantum examples. Then $D = O(nQ)$.*

These results draw on lower bound techniques from both quantum computation and computational learning theory [2, 5, 6, 8, 12, 26]. A detailed description of the relationship between our results and previous work on quantum versus classical black-box query complexity is given in section 3.4.

Theorems 1 and 2 are information-theoretic rather than computational in nature; they show that for any learning problem, if there is a quantum learning algorithm which uses polynomially many examples, then there must also exist a classical learning algorithm which uses polynomially many examples. However, Theorems 1 and 2 do not imply that every *polynomial-time* quantum learning algorithm must have a polynomial-time classical analogue. In fact, we show that a separation exists between efficient quantum learnability and efficient classical learnability. Under a widely held computational hardness assumption for classical computation (the hardness of factoring Blum integers), we observe that for each of the two learning models considered in this paper there is a concept class which is polynomial-time learnable in the quantum version but not in the classical version of the model.

For the model of exact learning from membership queries we give an even stronger separation between efficient quantum and classical learnability. Our third main theorem is the following.

THEOREM 3. *If any one-way function exists, then there is a concept class C which is polynomial-time exact learnable from quantum membership queries but is not polynomial-time exact learnable from classical membership queries.*

This result establishes a robust separation between efficient quantum and classical learnability. Even if a polynomial-time classical factoring algorithm were to be discovered, the separation would hold as long as *any* one-way function existed (a universally held belief in public-key cryptography). As discussed in section 9, our results prove the existence of a quantum oracle algorithm which defeats a general cryptographic construction secure in the classical setting. More precisely, given any one-way function, we construct a family of pseudorandom functions which are classically secure but can be distinguished from truly random functions (and in fact exactly identified) by an algorithm that can make quantum oracle queries. To our knowledge, this is the first break of a generic cryptographic construction (not based on a specific assumption such as factoring) in a quantum setting.

The main cryptographic tool underlying Theorem 3 is a new construction of pseudorandom functions which are invariant under an XOR mask (see section 7). As described in section 8.1, each concept $c \in C$ combines these new pseudorandom functions with pseudorandom permutations in a particular way. Roughly speaking, the XOR mask invariance of the new pseudorandom functions ensures that a quantum algorithm due to Simon [40] can be used to extract some information about the structure of the target concept and thus make progress towards learning. On the other hand, the pseudorandomness ensures that no probabilistic polynomial-time learning algorithm can extract any useful information, and thus no such algorithm can learn successfully.

1.3. Organization. We set notation, define the exact learning model and the PAC learning model, and describe the quantum computation framework in section 2. We prove the relationship between quantum and classical exact learning from membership queries (Theorem 1) in section 3, and we prove the relationship between quantum and classical PAC learning (Theorem 2) in section 4. In section 5 we observe that if factoring Blum integers is classically hard, then in each of these two learning models there is a concept class which is quantum learnable in polynomial time but not classically learnable in polynomial time. We prove Theorem 3 in sections 6 through 8.

2. Preliminaries. For $\alpha, \beta \in \{0,1\}$ we write $\alpha \oplus \beta$ to denote the exclusive-or $\alpha + \beta \pmod 2$. Similarly for $x, y \in \{0,1\}^n$ we write $x \oplus y$ to denote the n-bit string that is the bitwise XOR of x and y. We write $x \cdot y$ to denote the inner product $x_1 y_1 + \cdots + x_n y_n \pmod 2$, and we write $|x|$ to denote the length of string x.

We use script capital letters to denote probability distributions over sets of functions; in particular, $\mathcal{F}_n$ denotes the uniform distribution over all 2^{n2^n} functions from $\{0,1\}^n$ to $\{0,1\}^n$. If S is a finite set, we write $\Pr_{s \in S}$ to denote a uniform choice of s from S.

We write $M(s)$ to indicate that algorithm M is given string s as input, and M^g to indicate that M has access to an oracle for the function g. If M is a probabilistic polynomial-time (henceforth abbreviated p.p.t.) algorithm which has access to an oracle $g : \{0,1\}^{\ell_1} \to \{0,1\}^{\ell_2}$, then the running time of M^g is bounded by $p(\ell_1 + \ell_2)$ for some polynomial p.

A *concept* c over $\{0,1\}^n$ is a Boolean function over the domain $\{0,1\}^n$, or equivalently a concept can be viewed as a subset $\{x \in \{0,1\}^n : c(x) = 1\}$ of $\{0,1\}^n$. A *concept class* $\mathcal{C} = \cup_{n \geq 1} C_n$ is a collection of concepts, where $C_n = \{c \in \mathcal{C} : \ c \text{ is a}$

concept over $\{0,1\}^n\}$. For example, C_n might be the family of all Boolean formulae over n variables which are of size at most n^2. We say that a pair $\langle x, c(x)\rangle$ is a *labeled example* of the concept c.

While many different learning models have been proposed, most models follow the same basic paradigm: a learning algorithm for a concept class $\mathcal{C}$ typically has access to some kind of oracle which provides examples that are labeled according to a fixed but unknown target concept $c \in \mathcal{C}$, and the goal of the learning algorithm is to infer (in some sense) the target concept c. The two learning models which we discuss in this paper, the model of exact learning from membership queries and the PAC model, make this rough notion precise in different ways.

2.1. Classical exact learning from membership queries. The model of *exact learning from membership queries* was introduced by Angluin [2] and has since been widely studied [2, 12, 22, 26, 27]. In this model the learning algorithm has access to a *membership oracle* MQ_c, where $c \in C_n$ is the unknown target concept. When given an input string $x \in \{0,1\}^n$, in one time step the oracle MQ_c returns the bit $c(x)$; such an invocation is known as a *membership query* since the oracle's answer tells whether or not $x \in c$ (viewing c as a subset of $\{0,1\}^n$). The goal of the learning algorithm is to construct a hypothesis $h : \{0,1\}^n \to \{0,1\}$ that is logically equivalent to c, i.e., $h(x) = c(x)$ for all $x \in \{0,1\}^n$. Formally, we say that an algorithm A is an *exact learning algorithm for $\mathcal{C}$ using membership queries* if for all $n \geq 1$, for all $c \in C_n$, if A is given n and access to MQ_c, then with probability at least $2/3$ (over the internal randomness of A) algorithm A outputs a Boolean circuit h such that $h(x) = c(x)$ for all $x \in \{0,1\}^n$. The *sample complexity* $T(n)$ of a learning algorithm A for $\mathcal{C}$ is the maximum number of calls to MQ_c which A ever makes for any $c \in C_n$.

2.2. Classical PAC learning. The PAC model of concept learning was introduced by Valiant in [41] and has since been extensively studied [4, 29]. In this model the learning algorithm has access to an *example oracle* $EX(c, \mathcal{D})$, where $c \in C_n$ is the unknown target concept and $\mathcal{D}$ is an unknown distribution over $\{0,1\}^n$. The oracle $EX(c, \mathcal{D})$ takes no inputs; when invoked, in one time step it returns a labeled example $\langle x, c(x)\rangle$, where $x \in \{0,1\}^n$ is randomly selected according to the distribution $\mathcal{D}$. The goal of the learning algorithm is to generate a hypothesis $h : \{0,1\}^n \to \{0,1\}$ that is an *ϵ-approximator for c under $\mathcal{D}$*, i.e., a hypothesis h such that $\Pr_{x\in\mathcal{D}}[h(x) \neq c(x)] \leq \epsilon$. An algorithm A is a *PAC learning algorithm for $\mathcal{C}$* if the following condition holds: for all $n \geq 1$ and $0 < \epsilon, \delta < 1$, for all $c \in C_n$, for all distributions $\mathcal{D}$ over $\{0,1\}^n$, if A is given n, ϵ, δ and access to $EX(c, \mathcal{D})$, then with probability at least $1 - \delta$ algorithm A outputs a circuit h which is an ϵ-approximator for c under $\mathcal{D}$. The *sample complexity* $T(n, \epsilon, \delta)$ of a learning algorithm A for $\mathcal{C}$ is the maximum number of calls to $EX(c, \mathcal{D})$ that A ever makes for any concept $c \in C_n$ and any distribution $\mathcal{D}$ over $\{0,1\}^n$.

2.3. Quantum computation. Detailed descriptions of the quantum computation model can be found in [7, 16, 31, 45]; here we outline only the basics using the terminology of *quantum networks* as presented in [5]. A quantum network $\mathcal{N}$ is a quantum circuit (over some standard basis augmented with one oracle gate) that acts on an m-bit quantum register; the computational basis states of this register are the 2^m binary strings of length m. A quantum network can be viewed as a sequence of unitary transformations

$$U_0, O_1, U_1, O_2, \ldots, U_{T-1}, O_T, U_T,$$

where each U_i is an arbitrary unitary transformation on m qubits and each O_i is a unitary transformation which corresponds to an oracle call.[1] Such a network is said to have *query complexity* T. At every stage in the execution of the network, the current state of the register can be represented as a superposition $\sum_{z\in\{0,1\}^m} \alpha_z|z\rangle$, where the α_z are complex numbers that satisfy $\sum_{z\in\{0,1\}^m} \|\alpha_z\|^2 = 1$. If this state is measured, then with probability $\|\alpha_z\|^2$ the string $z \in \{0,1\}^m$ is observed and the state collapses down to $|z\rangle$. After the final transformation U_T takes place, a measurement is performed on some subset of the bits in the register, and the observed value (a classical bit string) is the output of the computation.

Several points deserve mention here. First, since the information which our quantum network uses for its computation comes from the oracle calls, we may stipulate that the initial state of the quantum register is $|0^m\rangle$. Second, as described above, each U_i can be an arbitrarily complicated unitary transformation (as long as it does not contain any oracle calls) which may require a large quantum circuit to implement. This is of small concern since we are chiefly interested in query complexity and not circuit size. Third, as defined above, our quantum networks can make only one measurement at the very end of the computation; this is an inessential restriction since any algorithm which uses intermediate measurements can be modified to an algorithm which makes only one final measurement. Finally, we have not specified just how the oracle calls O_i work; we address this point separately in sections 3.1 and 4.1 for each type of oracle.

If $|\phi\rangle = \sum_z \alpha_z|z\rangle$ and $|\psi\rangle = \sum_z \beta_z|z\rangle$ are two superpositions of basis states, then the *Euclidean distance* between $|\phi\rangle$ and $|\psi\rangle$ is $||\phi\rangle - |\psi\rangle| = (\sum_z |\alpha_z - \beta_z|^2)^{1/2}$. The *total variation distance* between two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ is defined to be $\sum_x |\mathcal{D}_1(x) - \mathcal{D}_2(x)|$. The following fact (Lemma 3.2.6 of [7]), which relates the Euclidean distance between two superpositions and the total variation distance between the distributions induced by measuring the two superpositions, will be useful.

FACT 1. *Let $|\phi\rangle$ and $|\psi\rangle$ be two unit-length superpositions which represent possible states of a quantum register. If the Euclidean distance $||\phi\rangle - |\psi\rangle|$ is at most ϵ, then performing the same observation on $|\phi\rangle$ and $|\psi\rangle$ induces distributions $\mathcal{D}_\phi$ and $\mathcal{D}_\psi$, which have total variation distance at most 4ϵ.*

3. Exact learning from quantum membership queries.

3.1. Quantum membership queries. A *quantum membership oracle* QMQ_c is the natural quantum generalization of a classical membership oracle MQ_c: on input of a superposition of query strings, the oracle QMQ_c generates the corresponding superposition of example labels. More formally, a QMQ_c gate maps the basis state $|x, b\rangle$ (where $x \in \{0,1\}^n$ and $b \in \{0,1\}$) to the state $|x, b\oplus c(x)\rangle$. If $\mathcal{N}$ is a quantum network which has QMQ_c gates as its oracle gates, then each O_i is the unitary transformation which maps $|x, b, y\rangle$ (where $x \in \{0,1\}^n$, $b \in \{0,1\}$ and $y \in \{0,1\}^{m-n-1}$) to $|x, b\oplus c(x), y\rangle$.[2] Our QMQ_c oracle is identical to the well-studied notion of a quantum black-box oracle for c [5, 6, 7, 9, 10, 11, 15, 17, 24, 46].

A *quantum exact learning algorithm for* $\mathcal{C}$ is a family $\mathcal{N}_1, \mathcal{N}_2, \ldots$ of quantum networks, where each network $\mathcal{N}_n$ has a fixed architecture independent of the choice of $c \in C_n$, with the following property: for all $n \geq 1$, for all $c \in C_n$, if the oracle gates of $\mathcal{N}_n$ are instantiated as QMQ_c gates, then with probability at least $2/3$ the network

[1] Since there is only one kind of oracle gate, each O_i is the same transformation.

[2] Note that each O_i affects only the first $n+1$ bits of a basis state. This is without loss of generality since the transformations U_j can "permute bits" of the network.

$\mathcal{N}_n$ outputs a representation of a (classical) Boolean circuit $h : \{0,1\}^n \to \{0,1\}$ such that $h(x) = c(x)$ for all $x \in \{0,1\}^n$. The *quantum sample complexity* of a quantum exact learning algorithm for $\mathcal{C}$ is $T(n)$, where $T(n)$ is the query complexity of $\mathcal{N}_n$.

3.2. Lower bounds on classical and quantum exact learning. Two different lower bounds are known for the number of classical membership queries which are required to exact learn any concept class. In this section we prove two analogous lower bounds on the number of quantum membership queries required to exactly learn any concept class. Throughout this section for ease of notation we omit the subscript n and write C for C_n.

A lower bound based on similarity of concepts. Consider a set of concepts which are all "similar" in the sense that for every input almost all concepts in the set agree. Known results in learning theory state that such a concept class must require a large number of membership queries for exact learning. More formally, let $C' \subseteq C$ be any subset of C. For $a \in \{0,1\}^n$ and $b \in \{0,1\}$ let $C'_{\langle a,b \rangle}$ denote the set of those concepts in C' which assign the label b to example a, i.e., $C'_{\langle a,b \rangle} = \{c \in C' : c(a) = b\}$. Let $\gamma^{C'}_{\langle a,b \rangle} = |C'_{\langle a,b \rangle}|/|C'|$ be the fraction of such concepts in C', and let $\gamma^{C'}_a = \min\{\gamma^{C'}_{\langle a,0 \rangle}, \gamma^{C'}_{\langle a,1 \rangle}\}$; thus $\gamma^{C'}_a$ is the minimum fraction of concepts in C' which can be eliminated by querying MQ_c on the string a. Let $\gamma^{C'} = \max\{\gamma^{C'}_a : a \in \{0,1\}^n\}$. Finally, let $\hat{\gamma}^C$ be the minimum of $\gamma^{C'}$ across all $C' \subseteq C$ such that $|C'| \geq 2$. Thus

$$\hat{\gamma}^C = \min_{C' \subseteq C, |C'| \geq 2} \max_{a \in \{0,1\}^n} \min_{b \in \{0,1\}} \frac{|C'_{\langle a,b \rangle}|}{|C'|}.$$

Intuitively, the inner min corresponds to the fact that the oracle may provide a worst-case response to any query; the max corresponds to the fact that the learning algorithm gets to choose the "best" query point a; and the outer min corresponds to the fact that the learner must succeed, no matter what subset C' of C the target concept is drawn from. Thus $\hat{\gamma}^C$ is small if there is a large set C' of concepts which are all very similar in that any query eliminates only a few concepts from C'. If this is the case, then many membership queries should be required to learn C; formally, we have the following lemma, which is a variant of Fact 2 from [12].

LEMMA 4. *Any (classical) exact learning algorithm for C must have sample complexity $\Omega(1/\hat{\gamma^C})$.*

Proof. Let $C' \subseteq C$, $|C'| \geq 2$, be such that $\gamma^{C'} = \hat{\gamma}^C$. Consider the following adversarial strategy for answering queries: given the query string a, answer the bit b which maximizes $\gamma^{C'}_{\langle a,b \rangle}$. This strategy ensures that each response eliminates at most a $\gamma^{C'}_a \leq \gamma^{C'} = \hat{\gamma}^C$ fraction of the concepts in C'. After $\frac{1}{2\hat{\gamma}^C} - 1$ membership queries, fewer than half of the concepts in C' have been eliminated, so at least two concepts have not yet been eliminated. Consequently, it is impossible for A to output a hypothesis which is equivalent to the correct concept with probability greater than $1/2$. ∎

We now develop some tools which will enable us to prove a quantum version of Lemma 4. Let $C' \subseteq C, |C'| \geq 2$, be such that $\gamma^{C'} = \hat{\gamma}^C$, and let $c_1, \ldots, c_{|C'|}$ be a listing of the concepts in C'. Let the *typical concept for C'* be the function $\hat{c} : \{0,1\}^n \to \{0,1\}$ defined as follows: for all $a \in \{0,1\}^n$, $\hat{c}(a)$ is the bit b such that $|C'_{\langle a,b \rangle}| \geq |C'|/2$. (Ties are broken arbitrarily; note that a tie occurs only if $\hat{\gamma}^C = 1/2$.) The typical concept $\hat{c}$ need not belong to C' or even to C. The *difference matrix* D is the $|C'| \times 2^n$ zero/one matrix where rows are indexed by concepts in C', columns are indexed by strings in $\{0,1\}^n$, and $D_{i,x} = 1$ iff $c_i(x) \neq \hat{c}(x)$. By our choice of C' and

the definition of $\hat{\gamma}^C$, each column of D has at most $|C'| \cdot \hat{\gamma}^C$ ones, so the L_1 matrix norm of D is $\|D\|_1 \leq |C'| \cdot \hat{\gamma}^C$.

Our quantum lower bound proof uses ideas which were first introduced by Bennett et al. [6]. Let $\mathcal{N}$ be a fixed quantum network architecture and let $U_0, O_1, \ldots, U_{T-1}$, O_T, U_T be the corresponding sequence of transformations. For $1 \leq t \leq T$ let $|\phi_t^c\rangle$ be the state of the quantum register after the transformations up through U_{t-1} have been performed (we refer to this stage of the computation as time t) if the oracle gate is QMQ_c. As in [6], for $x \in \{0,1\}^n$ let $q_x(|\phi_t^c\rangle)$, the *query magnitude of string x at time t with respect to c,* be the sum of the squared magnitudes in $|\phi_t^c\rangle$ of the basis states which are querying QMQ_c on string x at time t; thus if $|\phi_t^c\rangle = \sum_{z \in \{0,1\}^m} \alpha_z |z\rangle$, then

$$q_x(|\phi_t^c\rangle) = \sum_{w \in \{0,1\}^{m-n}} \|\alpha_{xw}\|^2.$$

The quantity $q_x(|\phi_t^c\rangle)$ can be viewed as the amount of amplitude which the network $\mathcal{N}$ invests in the query string x to QMQ_c at time t. Intuitively, the final outcome of $\mathcal{N}$'s computation cannot depend very much on the oracle's responses to queries which have little amplitude invested in them. Bennett et al. formalized this intuition in the following theorem [6, Theorem 3.3].

THEOREM 5. *Let $|\phi_t^c\rangle$ be defined as above. Let $F \subseteq \{0, \ldots, T-1\} \times \{0,1\}^n$ be a set of time-string pairs such that $\sum_{(t,x) \in F} q_x(|\phi_t^c\rangle) \leq \frac{\epsilon^2}{T}$. Now suppose that the answer to each query instance $(t,x) \in F$ is modified to some arbitrary fixed bit $a_{t,x}$ (these answers need not be consistent with any oracle). Let $|\tilde{\phi}_t^c\rangle$ be the state of the quantum register at time t if the oracle responses are modified as stated above. Then $||\phi_T^c\rangle - |\tilde{\phi}_T^c\rangle| \leq \epsilon$.*

The following lemma, which is an extension of Corollary 3.4 from [6], shows that no quantum learning algorithm that makes few QMQ queries can effectively distinguish many concepts in C' from the typical concept $\hat{c}$.

LEMMA 6. *Fix any quantum network architecture $\mathcal{N}$ which has query complexity T. For all $\epsilon > 0$ there is a set $S \subseteq C'$ of cardinality at most $T^2|C'|\hat{\gamma}^C/\epsilon^2$ such that for all $c \in C' \setminus S$ we have $||\phi_T^{\hat{c}}\rangle - |\phi_T^c\rangle| \leq \epsilon$.*

Proof. Since $||\phi_t^{\hat{c}}\rangle| = 1$ for $t = 0, 1, \ldots, T-1$, we have $\sum_{t=0}^{T-1} \sum_{x \in \{0,1\}^n} q_x(|\phi_t^{\hat{c}}\rangle) = T$. Let $q(|\phi_t^{\hat{c}}\rangle) \in \Re^{2^n}$ be the 2^n-dimensional vector which has entries indexed by strings $x \in \{0,1\}^n$ and which has $q_x(|\phi_t^{\hat{c}}\rangle)$ as its xth entry. Note that the L_1 norm $\|q(|\phi_t^{\hat{c}}\rangle)\|_1$ is 1 for all $t = 0, \ldots, T-1$. For any $c_i \in C'$ let $q_{c_i}(|\phi_t^{\hat{c}}\rangle)$ be defined as $\sum_{x : c_i(x) \neq \hat{c}(x)} q_x(|\phi_t^{\hat{c}}\rangle)$. The quantity $q_{c_i}(|\phi_t^{\hat{c}}\rangle)$ can be viewed as the total query magnitude with respect to $\hat{c}$ at time t of those strings which distinguish c_i from $\hat{c}$. Note that $Dq(|\phi_t^{\hat{c}}\rangle) \in \Re^{|C'|}$ is a $|C'|$-dimensional vector whose ith element is precisely $\sum_{x : c_i(x) \neq \hat{c}(x)} q_x(|\phi_t^{\hat{c}}\rangle) = q_{c_i}(|\phi_t^{\hat{c}}\rangle)$. Since $\|D\|_1 \leq |C'| \cdot \hat{\gamma}^C$ and $\|q(|\phi_t^{\hat{c}}\rangle)\|_1 = 1$, by the basic property of matrix norms we have that $\|Dq(|\phi_t^{\hat{c}}\rangle)\|_1 \leq |C'| \cdot \hat{\gamma}^C$, i.e., $\sum_{c_i \in C'} q_{c_i}(|\phi_t^{\hat{c}}\rangle) \leq |C'| \cdot \hat{\gamma}^C$. Hence

$$\sum_{t=0}^{T-1} \sum_{c_i \in C'} q_{c_i}(|\phi_t^{\hat{c}}\rangle) \leq T|C'| \cdot \hat{\gamma}^C.$$

If we let $S = \{c_i \in C' : \sum_{t=0}^{T-1} q_{c_i}(|\phi_t^{\hat{c}}\rangle) \geq \frac{\epsilon^2}{T}\}$, by Markov's inequality we have $|S| \leq T^2|C'|\hat{\gamma}^C/\epsilon^2$. Finally, if $c \notin S$, then $\sum_{t=0}^{T-1} q_c(|\phi_t^{\hat{c}}\rangle) \leq \frac{\epsilon^2}{T}$. Theorem 5 then implies that $||\phi_T^{\hat{c}}\rangle - |\phi_T^c\rangle| \leq \epsilon$. ∎

Now we can prove our quantum version of Lemma 4.

THEOREM 7. *Any quantum exact learning algorithm for C must have sample complexity* $\Omega\Big(\big(1/\hat{\gamma}^C\big)^{1/2}\Big)$.

Proof. Suppose that $\mathcal{N}$ is a quantum exact learning algorithm for $\mathcal{C}$ which makes at most $T = \frac{1}{64} \cdot (\frac{1}{\hat{\gamma}^C})^{1/2}$ quantum membership queries. If we take $\epsilon = \frac{1}{32}$, then Lemma 6 implies that there is a set $S \subset C'$ of cardinality at most $\frac{|C'|}{4}$ such that for all $c \in C' \setminus S$ we have $||\phi_T^c\rangle - |\phi_T^{\hat{c}}\rangle| \leq \frac{1}{32}$. Let c_1, c_2 be any two concepts in $C' \setminus S$. By Fact 1, the probability that $\mathcal{N}$ outputs a circuit equivalent to c_1 can differ by at most $\frac{1}{8}$ if $\mathcal{N}$'s oracle gates are $QMQ_{\hat{c}}$ as opposed to QMQ_{c_1}, and likewise for $QMQ_{\hat{c}}$ versus QMQ_{c_2}. It follows that the probability that $\mathcal{N}$ outputs a circuit equivalent to c_1 can differ by at most one fourth if $\mathcal{N}$'s oracle gates are QMQ_{c_1} as opposed to QMQ_{c_2}, but this contradicts the assumption that $\mathcal{N}$ is a quantum exact learning algorithm for C. □

Well-known results [9, 24] show that $O(\sqrt{N})$ queries are sufficient to search a quantum database of N unordered items for a desired item. These upper bounds can easily be used to show that Theorem 7 is tight up to constant factors.

A lower bound based on concept class size. A second reason why a concept class can require many membership queries is its size. Angluin [2] has given the following simple bound, incomparable to the bound of Lemma 4, on the number of classical membership queries required for exact learning.

LEMMA 8. *Any classical exact learning algorithm for C must have sample complexity* $\Omega(\log |C|)$.

Proof. Consider the following adversarial strategy for answering queries: if $C' \subseteq C$ is the set of concepts which have not yet been eliminated by previous responses to queries, then given the query string a, answer the bit b such that $\gamma^{C'}_{\langle a,b\rangle} \geq \frac{1}{2}$. Under this strategy, after $\log |C| - 1$ membership queries, at least two possible target concepts will remain. □

In this section we prove a variant of this lemma for the quantum model. Our proof uses ideas from [5], so we introduce some of their notation. Let $N = 2^n$. For each concept $c \in C$, let $X^c = (X^c_0, \ldots, X^c_{N-1}) \in \{0,1\}^N$ be a vector which represents c as an N-tuple, i.e., $X^c_i = c(x^i)$, where $x^i \in \{0,1\}^n$ is the binary representation of i. From this perspective we may identify C with a subset of $\{0,1\}^N$, and we may view a QMQ_c gate as a black-box oracle for X^c which maps basis state $|x^i, b, y\rangle$ to $|x^i, b \oplus X^c_i, y\rangle$.

Using ideas from [20, 21], Beals et al. have proved the following useful lemma, which relates the query complexity of a quantum network to the degree of a certain polynomial [5, Lemma 4.2].

LEMMA 9. *Let $\mathcal{N}$ be a quantum network that makes T queries to a black-box X, and let $B \subseteq \{0,1\}^m$ be a set of basis states. Then there exists a real-valued multilinear polynomial $P_B(X)$ of degree at most $2T$ which equals the probability that observing the final state of the network with black-box X yields a state from B.*

We use Lemma 9 to prove the following quantum lower bound based on concept class size. (de Wolf has observed [18] that this lower bound can also be obtained from the results of [19].)

THEOREM 10. *Any exact quantum learning algorithm for C must have sample complexity* $\Omega\big(\frac{\log |C|}{n}\big)$.

Proof. Let $\mathcal{N}$ be a quantum network which learns C and has query complexity T. For all $c \in C$ we have the following: if $\mathcal{N}$'s oracle gates are QMQ_c gates, then

with probability at least 2/3 the output of $\mathcal{N}$ is a representation of a Boolean circuit h which computes c. Let $c_1, \ldots, c_{|C|}$ be all of the concepts in C, and let $X^1, \ldots, X^{|C|}$ be the corresponding vectors in $\{0,1\}^N$. For all $i = 1, \ldots, |C|$ let $B_i \subseteq \{0,1\}^m$ be the collection of those basis states which are such that if the final observation performed by $\mathcal{N}$ yields a state from B_i, then the output of $\mathcal{N}$ is a representation of a Boolean circuit which computes c_i. Clearly for $i \neq j$ the sets B_i and B_j are disjoint. By Lemma 9, for each $i = 1, \ldots, |C|$ there is a real-valued multilinear polynomial P_i of degree at most $2T$ such that for all $j = 1, \ldots, |C|$ the value of $P_i(X^j)$ is precisely the probability that the final observation on $\mathcal{N}$ yields a representation of a circuit which computes c_i, provided that the oracle gates are QMQ_{c_j} gates. The polynomials P_i thus have the following properties:

1. $P_i(X^i) \geq 2/3$ for all $i = 1, \ldots, |C|$;
2. for any $j = 1, \ldots, |C|$, we have $\sum_{i \neq j} P_i(X^j) \leq 1/3$ (since the total probability across all possible observations is 1).

Let $N_0 = \sum_{i=0}^{2T} \binom{N}{i}$. For any $X = (X_0, \ldots, X_{N-1}) \in \{0,1\}^N$ let $\tilde{X} \in \{0,1\}^{N_0}$ be the column vector which has a coordinate for each monic multilinear monomial over $X_0, \ldots, X_{N-1}$ of degree at most $2T$. Thus, for example, if $N = 4$ and $2T = 2$, we have $X = (X_0, X_1, X_2, X_3)$ and

$$\tilde{X}^t = (1, X_0, X_1, X_2, X_3, X_0X_1, X_0X_2, X_0X_3, X_1X_2, X_1X_3, X_2X_3).$$

If V is a column vector in $\Re^{N_0}$, then $V^t\tilde{X}$ corresponds to the degree-$2T$ polynomial whose coefficients are given by the entries of V. For $i = 1, \ldots, |C|$ let $V_i \in \Re^{N_0}$ be the column vector which corresponds to the coefficients of the polynomial P_i. Let M be the $|C| \times N_0$ matrix whose ith row is V_i^t; note that multiplication by M defines a linear transformation from $\Re^{N_0}$ to $\Re^{|C|}$. Since $V_i^t\tilde{X}^j$ is precisely $P_i(X^j)$, the product $M\tilde{X}^j$ is a column vector in $\Re^{|C|}$ which has $P_i(X^j)$ as its ith coordinate.

Now let L be the $|C| \times |C|$ matrix whose jth column is the vector $M\tilde{X}^j$. A square matrix A is said to be *diagonally dominant* if $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ for all i. Properties 1 and 2 above imply that the transpose of L is diagonally dominant. It is well known that any diagonally dominant matrix must be of full rank (see, e.g., [32]). Since L is of full rank and each column of L is in the image of M, it follows that the image under M of $\Re^{N_0}$ is all of $\Re^{|C|}$, and hence $N_0 \geq |C|$. Finally, since $N_0 = \sum_{i=0}^{2T} \binom{N}{i} \leq N^{2T}$, we have $T \geq \frac{\log |C|}{2 \log N} = \frac{\log |C|}{2n}$, which proves the theorem. □

The lower bound of Theorem 10 is nearly tight, as witnessed by the following example: let C be the collection of all 2^n parity functions over $\{0,1\}^n$, so that each function in C is defined by a string $a \in \{0,1\}^n$ and $c_a(x) = a \cdot x$. The quantum algorithm which solves the well-known Deutsch–Jozsa problem [17] can be used to exactly identify a and thus learn the target concept with probability 1 from a single query. It follows that the factor of n in the denominator of Theorem 10 cannot be replaced by any function $g(n) = o(n)$.

3.3. Quantum and classical exact learning are equivalent. We have seen two different reasons why exact learning of a concept class can require a large number of classical membership queries: the class may contain many similar concepts (i.e., $\hat{\gamma}^C$ is small) or the class may contain very many concepts (i.e., $\log |C|$ is large). The following lemma, which is a variant of Theorem 3.1 from [26], shows that these are the only reasons why many membership queries may be required.

LEMMA 11. *There is an exact learning algorithm for C which has sample complexity $O((\log |C|)/\hat{\gamma}^C)$.*

Proof. Consider the following learning algorithm A: at each stage in its execution, if C' is the set of concepts in C which have not yet been eliminated by previous responses to queries, algorithm A's next query string is the string $a \in \{0,1\}^n$, which maximizes $\gamma_a^{C'}$. By following this strategy, each query response received from the oracle must eliminate at least a $\gamma^{C'}$ fraction of the set C', so with each query the size of the set of possible target concepts is multiplied by a factor which is at most $1 - \gamma^{C'} \leq 1 - \hat{\gamma}^C$. Consequently, after $O((\log |C|)/\hat{\gamma}^C)$ queries, only a single concept will not have been eliminated; this concept must be the target concept, so that A can output a hypothesis h which is equivalent to c. ∎

Combining Theorem 7, Theorem 10, and Lemma 11, we obtain the following relationship between the quantum and classical sample complexity of exact learning.

THEOREM 12. *Let $\mathcal{C}$ be any concept class over $\{0,1\}^n$, and let D and Q be such that $\mathcal{C}$ is exact learnable from D classical membership queries or from Q quantum membership queries. Then $D = O(nQ^3)$.*

We note that a QMQ_c oracle can clearly be used to simulate an MQ_c oracle, and thus $Q \leq D$ as well.

3.4. Discussion. Theorem 12 provides an interesting contrast to several known results for black-box quantum computation. Let F denote the set of all 2^{2^n} functions from $\{0,1\}^n$ to $\{0,1\}$. Beals et al. [5] have shown that if $f : F \to \{0,1\}$ is any total function (i.e., $f(c)$ is defined for every possible concept c over $\{0,1\}^n$), then the query complexity of any quantum network that computes f is polynomially related to the number of classical black-box queries required to compute f. Their result is interesting because it is well known [7, 11, 17, 40] that, for certain concept classes $C \subset F$ and partial functions $f : C \to \{0,1\}$, the quantum black-box query complexity of f can be exponentially smaller than the classical black-box query complexity.

Our Theorem 12 provides a sort of dual to the results of Beals et al.: their bound on query complexity holds only for the fixed concept class F but for any function $f : F \to \{0,1\}$, while our bound holds for any concept class $C \subseteq F$ but only for the fixed problem of exact learning. In general, the problem of computing a function $f : C \to \{0,1\}$ from black-box queries can be viewed as an easier version of the corresponding exact learning problem: instead of having to figure out only one bit of information about the unknown concept c (the value of f), for the learning problem the algorithm must identify c exactly. Theorem 1 shows that for this more demanding problem, unlike the results in [7, 11, 17, 40], there is no way of restricting the concept class C so that learning becomes substantially easier in the quantum setting than in the classical setting.

4. PAC learning from a quantum example oracle.

4.1. The quantum example oracle. Bshouty and Jackson [13] have introduced a natural quantum generalization of the standard PAC-model example oracle. While a standard PAC example oracle $EX(c, \mathcal{D})$ generates each example $\langle x, c(x)\rangle$ with probability $\mathcal{D}(x)$, where $\mathcal{D}$ is a distribution over $\{0,1\}^n$, a *quantum PAC example oracle* $QEX(c, \mathcal{D})$ generates a superposition of all labeled examples, where each labeled example $\langle x, c(x)\rangle$ appears in the superposition with amplitude proportional to the square root of $\mathcal{D}(x)$. More formally, a $QEX(c, \mathcal{D})$ gate maps the initial basis state $|0^n, 0\rangle$ to the state $\sum_{x \in \{0,1\}^n} \sqrt{\mathcal{D}(x)}|x, c(x)\rangle$. (We leave the action of a $QEX(c, \mathcal{D})$ gate undefined on other basis states, and stipulate that any quantum network which includes T $QEX(c, \mathcal{D})$ gates must have all T gates at the "bottom of the circuit," i.e., no gate may occur on any wire between the inputs and any $QEX(c, \mathcal{D})$ gate.) A

quantum network with T $QEX(c, \mathcal{D})$ gates is said to be a QEX network with *query complexity* T.

A *quantum PAC learning algorithm for* $\mathcal{C}$ is a family $\{\mathcal{N}_{(n,\epsilon,\delta)} : n \geq 1,\ 0 < \epsilon, \delta < 1\}$ of QEX networks with the following property: for all $n \geq 1$ and $0 < \epsilon, \delta < 1$, for all $c \in C_n$, for all distributions $\mathcal{D}$ over $\{0,1\}^n$, if the network $\mathcal{N}_{(n,\epsilon,\delta)}$ has all its oracle gates instantiated as $QEX(c, \mathcal{D})$ gates, then with probability at least $1 - \delta$ the network $\mathcal{N}_{(n,\epsilon,\delta)}$ outputs a representation of a circuit h, which is an ϵ-approximator to c under $\mathcal{D}$. The *quantum sample complexity* $T(n, \epsilon, \delta)$ of a quantum PAC algorithm is the query complexity of $\mathcal{N}_{(n,\epsilon,\delta)}$.

4.2. Lower bounds on classical and quantum PAC learning. Throughout this section for ease of notation we omit the subscript n and write C for C_n. We view each concept $c \in C$ as a subset of $\{0,1\}^n$. For $S \subseteq \{0,1\}^n$, we write $\Pi_C(S)$ to denote $\{c \cap S : c \in C\}$, so $|\Pi_C(S)|$ is the number of different "dichotomies" which the concepts in C induce on the points in S. A subset $S \subseteq \{0,1\}^n$ is said to be *shattered* by C if $|\Pi_C(S)| = 2^{|S|}$, i.e., if C induces every possible dichotomy on the points in S. The *Vapnik–Chervonenkis dimension of* C, VC-DIM(C), is the size of the largest subset $S \subseteq \{0,1\}^n$ which is shattered by C (see [43]).

Well-known results in computational learning theory show that the Vapnik–Chervonenkis dimension of a concept class C characterizes the number of calls to $EX(c, \mathcal{D})$, which are information-theoretically necessary and sufficient to PAC learn C. For the lower bound, the following theorem is a slight simplification of a result due to Blumer et al. [8, Theorem 2.1.ii.b].

THEOREM 13. *Let C be any concept class and $d =$ VC-DIM(C). Then any (classical) PAC learning algorithm for $\mathcal{C}$ must have sample complexity $\Omega(d)$.*

Proof sketch. The idea behind Theorem 13 is to consider the distribution $\mathcal{D}$ which is uniform over some shattered set S of size d and assigns zero weight to points outside of S. Any learning algorithm which makes only $d/2$ calls to $EX(c, \mathcal{D})$ will have no information about the value of c on at least $d/2$ points in S; moreover, since the set S is shattered by C, any labeling is possible for these unseen points. Since the error of any hypothesis h under $\mathcal{D}$ is the fraction of points in S at which h and the target concept disagree, a simple analysis shows that no learning algorithm which performs only $d/2$ calls to $EX(c, \mathcal{D})$ can have a high probability (e.g., $1 - \delta = 2/3$) of generating a low-error hypothesis (e.g., $\epsilon = 1/10$). □

We now give a quantum analogue of the classical lower bound given by Theorem 13.

THEOREM 14. *Let $\mathcal{C}$ be any concept class and $d =$ VC-DIM(C). Then any quantum PAC learning algorithm for $\mathcal{C}$ must have quantum sample complexity $\Omega(\frac{d}{n})$.*

Proof. Let $S = \{x^1, \ldots, x^d\}$ be a set which is shattered by C, and let $\mathcal{D}$ be the distribution which is uniform on S and assigns zero weight to points outside S. If $h : \{0,1\}^n \to \{0,1\}$ is a Boolean function on $\{0,1\}^n$, we say that the *relative distance of h and c on S* is the fraction of points in S at which h and c disagree. We will prove the following result, which is stronger than Theorem 14: let $\mathcal{N}$ be a quantum network with QMQ gates such that for all $c \in C$, if $\mathcal{N}$'s oracle gates are QMQ_c gates, then with probability at least $2/3$ the output of $\mathcal{N}$ is a hypothesis h such that the relative distance of h and c on S is at most $1/10$. We will show that such a network $\mathcal{N}$ must have query complexity at least $d/12n$. Since any QEX network with query complexity T can be simulated by a QMQ network with query complexity T, taking $\epsilon = 1/10$ and $\delta = 1/3$ will prove Theorem 14.

The argument is a modification of the proof of Theorem 10 using ideas from

error correcting codes. Let $\mathcal{N}$ be a quantum network with query complexity T which satisfies the following condition: for all $c \in C$, if $\mathcal{N}$'s oracle gates are QMQ_c gates, then with probability at least $2/3$ the output of $\mathcal{N}$ is a representation of a Boolean circuit h such that the relative distance of h and c on S is at most $1/10$. By the well-known Gilbert–Varshamov bound from coding theory (see, e.g., Theorem 5.1.7 of [42]), there exists a set $s^1, \ldots, s^A$ of d-bit strings such that for all $i \neq j$ the strings s^i and s^j differ in at least $d/4$ bit positions, where

$$A \geq \frac{2^d}{\sum_{i=0}^{d/4-1} \binom{d}{i}} \geq \frac{2^d}{\sum_{i=0}^{d/4} \binom{d}{i}} \geq 2^{d(1-H(1/4))} > 2^{d/6}.$$

(Here $H(p) = -p \log p - (1-p)\log(1-p)$ is the binary entropy function.) For each $i = 1, \ldots, A$ let $c_i \in C$ be a concept such that the d-bit string $c_i(x^1) \cdots c_i(x^d)$ is s^i (such a concept c_i must exist since the set S is shattered by C).

For $i = 1, \ldots, A$ let $B_i \subseteq \{0,1\}^m$ be the collection of those basis states which are such that if the final observation performed by $\mathcal{N}$ yields a state from B_i, then the output of $\mathcal{N}$ is a hypothesis h such that h and c_i have relative distance at most $1/10$ on S. Since each pair of concepts c_i, c_j has relative distance at least $1/4$ on S, the sets B_i and B_j are disjoint for all $i \neq j$.

As in section 3.2, let $N = 2^n$ and let $X^j = (X^j_0, \ldots, X^j_{N-1}) \in \{0,1\}^n$, where X^j is the N-tuple representation of the concept c_j. By Lemma 9, for each $i = 1, \ldots, A$ there is a real-valued multilinear polynomial P_i of degree at most $2T$ such that for all $j = 1, \ldots, A$ the value of $P_i(X^j)$ is precisely the probability that the final observation on $\mathcal{N}$ yields a state from B_i, provided that the oracle gates are QMQ_{c_j} gates. Since, by assumption, if c_i is the target concept, then with probability at least $2/3$, $\mathcal{N}$ generates a hypothesis that has relative distance at most $1/10$ from c_i on S, the polynomials P_i have the following properties:

1. $P_i(X^i) \geq 2/3$ for all $i = 1, \ldots, A$;
2. for any $j = 1, \ldots, A$ we have that $\sum_{i \neq j} P_i(X^j) \leq 1/3$ (since the B_i's are disjoint and the total probability across all observations is 1).

Let N_0 and $\tilde{X}$ be defined as in the proof of Theorem 10. For $i = 1, \ldots, A$ let $V_i \in \Re^{N_0}$ be the column vector which corresponds to the coefficients of the polynomial P_i, so that $V_i^t \tilde{X} = P_i(X)$. Let M be the $A \times N_0$ matrix whose ith row is the vector V_i^t, so that multiplication by M is a linear transformation from $\Re^{N_0}$ to $\Re^A$. The product $M\tilde{X}^j$ is a column vector in $\Re^A$ which has $P_i(X)$ as its ith coordinate.

Now let L be the $A \times A$ matrix whose jth column is the vector $M\tilde{X}^j$. As in Theorem 10, we have that the transpose of L is diagonally dominant, and thus L is of full rank and hence $N_0 \geq A$. Since $A \geq 2^{d/6}$, we thus have that $T \geq \frac{d/6}{2 \log_2 N} = \frac{d}{12n}$, and the theorem is proved. □

Since the class of parity functions over $\{0,1\}^n$ has VC-dimension n, as in Theorem 10 the n in the denominator of Theorem 14 cannot be replaced by any function $g(n) = o(n)$.

4.3. Quantum and classical PAC learning are equivalent. A well-known theorem due to Blumer et al. (Theorem 3.2.1.ii.a of [8]) shows that VC-DIM(C) also upper-bounds the number of $EX(c, \mathcal{D})$ calls required for (classical) PAC learning.

THEOREM 15. *Let C be any concept class and $d =$ VC-DIM(C). There is a classical PAC learning algorithm for C which has sample complexity $O(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{d}{\epsilon} \log \frac{1}{\epsilon})$.*

The proof of Theorem 15 is quite complex, and thus we do not attempt to sketch it. As in section 3.3, this upper bound along with our lower bound from Theorem 14

together yield the following.

THEOREM 16. *Let $\mathcal{C}$ be any concept class over $\{0,1\}^n$, and let D and Q be such that $\mathcal{C}$ is PAC learnable from D classical examples or from Q quantum examples. Then $D = O(nQ)$.*

We note that a $QEX(c, \mathcal{D})$ oracle can be used to simulate the corresponding $EX(c, \mathcal{D})$ oracle by immediately performing an observation on the QEX gate's outputs[3] (such an observation yields each example $\langle x, c(x)\rangle$ with probability $\mathcal{D}(x)$), and thus $Q \leq D$.

5. Quantum versus classical efficient learnability. We have shown that, from an information-theoretic perspective, up to polynomial factors quantum learning is no more powerful than classical learning. However, we now observe that the apparent *computational* advantages of the quantum model yield efficient quantum learning algorithms which seem to have no efficient classical counterparts.

A *Blum integer* is an integer $N = pq$, where $p \neq q$ are ℓ-bit primes each congruent to 3 modulo 4. It is widely believed that there is no polynomial-time classical algorithm which can successfully factor a randomly selected Blum integer with nonnegligible success probability.

Kearns and Valiant [28] have constructed a concept class $\mathcal{C}$ whose PAC learnability is closely related to the problem of factoring Blum integers. In their construction each concept $c \in \mathcal{C}$ is uniquely defined by some Blum integer N. Furthermore, c has the property that if $c(x) = 1$, then the prefix of x is the binary representation of N. Kearns and Valiant prove that if there is a polynomial-time PAC learning algorithm for $\mathcal{C}$, then there is a polynomial-time algorithm which factors Blum integers. Thus, assuming that factoring Blum integers is a computationally hard problem for classical computation, the Kearns–Valiant concept class $\mathcal{C}$ is not efficiently PAC learnable.

On the other hand, in a celebrated result, Shor [39] has exhibited a poly(n) size quantum network that can factor any n-bit integer with high success probability. Since each positive example of a concept $c \in \mathcal{C}$ reveals the Blum integer N which defines c, using Shor's algorithm it is easy to obtain an efficient quantum PAC learning algorithm for the Kearns–Valiant concept class. We thus have the following.

OBSERVATION 2. *If there is no polynomial-time classical algorithm for factoring Blum integers, then there is a concept class $\mathcal{C}$ that is efficiently quantum PAC learnable but not efficiently classically PAC learnable.*

The hardness results of Kearns and Valiant were later extended by Angluin and Kharitonov [3]. Using a public-key encryption system which is secure against chosen-cyphertext attack (based on the assumption that factoring Blum integers is computationally hard for polynomial-time algorithms), they constructed a concept class $\mathcal{C}$ which cannot be learned by any polynomial-time learning algorithm that makes membership queries. As with the Kearns–Valiant concept class, though, using Shor's quantum factoring algorithm it is possible to construct an efficient quantum exact learning algorithm for this concept class. Thus, for the exact learning model as well, we have the following.

OBSERVATION 3. *If there is no polynomial-time classical algorithm for factoring Blum integers, then there is a concept class $\mathcal{C}$ which is efficiently quantum exact learnable from membership queries but not efficiently classically exact learnable from membership queries.*

[3] As noted in section 2.3, intermediate observations during a computation can always be simulated by a single observation at the end of the computation.

In the next sections we prove Theorem 3, which establishes a stronger computational separation between the quantum and classical models of exact learning from membership queries than is implied by Observation 3. The proof involves Simon's quantum oracle algorithm, which we briefly describe in the next section.

6. Simon's algorithm. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function, and let $0^n \neq s \in \{0,1\}^n$. We say that f is *two-to-one with XOR mask* s if for all $y \neq x$, $f(x) = f(y) \iff y = x \oplus s$. More generally, f is *invariant under XOR mask with* s if $f(x) = f(x \oplus s)$ for all $x \in \{0,1\}^n$ (note that such a function need not be two-to-one).

Simon [40] has given a simple quantum algorithm which takes oracle access to a function $f : \{0,1\}^n \to \{0,1\}^n$, runs in poly$(n)$ time, and behaves as follows:

1. If f is a permutation on $\{0,1\}^n$, the algorithm outputs an n-bit string y which is uniformly distributed over $\{0,1\}^n$.
2. If f is two-to-one with XOR mask s, the algorithm outputs an n-bit string y which is uniformly distributed over the 2^{n-1} strings such that $y \cdot s = 0$.
3. If f is invariant under XOR mask with s, the algorithm outputs some n-bit string y which satisfies $y \cdot s = 0$.

Simon showed that by running this procedure $O(n)$ times, a quantum algorithm can distinguish between case 1 (f is a permutation) and case 3 (f is invariant under some XOR mask) with high probability. In case 1, after $O(n)$ repetitions the strings obtained will with probability $1 - 2^{-O(n)}$ contain a basis for the vector space $(Z_2)^n$ (here we are viewing n-bit strings as vectors over Z_2), while in case 3 the strings obtained cannot contain such a basis since each string must lie in the subspace $\{y : y \cdot s = 0\}$. Simon also observed that in case 2 (f is two-to-one with XOR mask s) the algorithm can be used to efficiently identify s with high probability. This is because after $O(n)$ repetitions, with high probability s will be the unique nonzero vector whose dot product with each y is 0; this vector can be found by solving the linear system defined by the y's.

Simon also analyzed the success probability of classical oracle algorithms for this problem. His analysis establishes the following theorem.

THEOREM 17. *Let $0^n \neq s \in \{0,1\}^n$ be chosen uniformly and let $f : \{0,1\}^n \to \{0,1\}^n$ be an oracle chosen uniformly from the set of all functions which are two-to-one with XOR mask s. Then* (i) *there is a polynomial-time quantum oracle algorithm which identifies s with high probability;* (ii) *any p.p.t. classical oracle algorithm identifies s with probability $1/2^{\Omega(n)}$.*

This surprising ability of quantum oracle algorithms to efficiently find s is highly suggestive in the context of our search for a learning problem which separates efficient classical and quantum computation. Indeed, Simon's algorithm will play a crucial role in establishing that the concept class which we construct in section 8 is learnable in poly(n) time by a quantum algorithm. Recall that in our learning scenario, though, the goal is to *exactly identify* the unknown target function, not just to identify the string s. Since $2^{\Omega(n)}$ bits are required to specify a randomly chosen function f which is two-to-one with XOR mask s, no algorithm (classical or quantum) can output a description of f in poly(n) time, much less learn f in poly(n) time. Thus it will not do to use truly random functions for our learning problem; instead we use *pseudorandom* functions as described in the next section.

7. Pseudorandomness. A *pseudorandom function family* [23] is a collection of functions $\{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ with the following two properties:

- *(efficient evaluation)* there is a deterministic algorithm which, given an n-bit seed s and an n-bit input x, runs in time poly(n) and outputs $f_s(x)$;

- *(pseudorandomness)* for all polynomials Q, all p.p.t. oracle algorithms M, and all sufficiently large n, we have that

$$\left| \Pr_{F \in \mathcal{F}_n} [M^F \text{ outputs } 1] - \Pr_{s \in \{0,1\}^n} [M^{f_s} \text{ outputs } 1] \right| < \frac{1}{Q(n)}.$$

Intuitively, the pseudorandomess property ensures that in any p.p.t. computation that uses a truly random function, a randomly chosen pseudorandom function may be used instead without affecting the outcome in a noticeable way. Well-known results [23, 25] imply that pseudorandom function families exist iff any one-way function exists.

A *pseudorandom permutation family* is a pseudorandom function family with the added property that each function $f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}$ is a permutation. Luby and Rackoff [30] gave the first construction of a pseudorandom permutation family from any pseudorandom function family. In their construction each permutation $f_s : \{0,1\}^n \to \{0,1\}^n$ has a seed s of length $|s| = 3n/2$ rather than n as in our definition above. Subsequent constructions [33, 34, 35] of pseudorandom permutation families $\{f_s : \{0,1\}^n \to \{0,1\}^n\}$ use n-bit seeds and hence match our definition exactly. (Our definition of pseudorandomness could easily be extended to allow seed lengths other than n. For our construction in section 8 it will be convenient to have n-bit seeds.)

7.1. Pseudorandom functions invariant under XOR mask. Our main cryptographic result, stated below, is proved in the appendix.

THEOREM 18. *If any one-way function exists, then there is a pseudorandom function family* $\{g_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}$ *such that* $g_s(x) = g_s(x \oplus s)$ *for all* $|x| = |s|$.

A first approach to constructing such a family is as follows: given any pseudorandom function family $\{f_s\}$, let $\{g_s\}$ be defined by

$$g_s(x) \stackrel{\text{def}}{=} f_s(x) \oplus f_s(x \oplus s). \tag{1}$$

This simple construction ensures that each function g_s is invariant under XOR mask with s, but the family $\{g_s\}$ need not be pseudorandom just because $\{f_s\}$ is pseudorandom. Indeed, if $\{h_s\}$ is similarly defined by $h_s(x) \stackrel{\text{def}}{=} g_s(x) \oplus g_s(x \oplus s)$, then $\{h_s\}$ is not pseudorandom since

$$h_s(x) \;=\; (f_s(x) \oplus f_s(x \oplus s)) \oplus (f_s(x \oplus s) \oplus f_s(x \oplus s \oplus s)) \;=\; 0^n.$$

While this example shows that (1) does not always preserve pseudorandomness, it leaves open the possibility that (1) may preserve pseudorandomness for certain function families $\{f_s\}$. In the appendix we show that if $\{f_s\}$ is a pseudorandom function family which is constructed from any one-way function in a particular way, then the family $\{g_s\}$ defined by (1) is indeed pseudorandom.

It may at first appear that the pseudorandom function family $\{g_s\}$ given by Theorem 18 immediately yields a concept class which separates efficient quantum learning from efficient classical learning. The pseudorandomness of $\{g_s\}$ ensures that no p.p.t. algorithm can learn successfully; on the other hand, if Simon's quantum algorithm is given oracle access to a function which is two-to-one with XOR mask s, then it can efficiently find s with high probability. Hence it may seem that, given access to g_s, Simon's quantum algorithm can efficiently identify the seed s and thus learn the target concept.

The flaw in this argument is that each function g_s from Theorem 18, while invariant under XOR mask with s, need not be two-to-one. Indeed g_s could conceivably be invariant under XOR mask with, say, $\sqrt{n}$ linearly independent strings $s = s^1, s^2, \ldots, s^{\sqrt{n}}$. Such a set of strings spans a $2^{\sqrt{n}}$-element subspace of $\{0,1\}^n$; even if Simon's algorithm could identify this subspace, it would not indicate which element of the subspace is the true seed s. Hence a more sophisticated construction is required.

8. Proof of Theorem 3.

8.1. The concept class C. We describe concepts over $\{0,1\}^m$, where $m = n + 2\log n + 1$. Each concept in C_m is defined by an $(n+1)$-tuple $(y, s^1, \ldots, s^n)$, where $y = y_1 \cdots y_n \in \{0,1\}^n$ and each $s^i \in \{0,1\}^n \setminus \{0^n\}$, so C_m contains $2^n(2^n-1)^n$ distinct concepts. For brevity we write $\tilde{s}$ to stand for $s^1, \ldots, s^n$ below.

Roughly speaking, each concept in C_m comprises n pseudorandom functions; as explained below, the string y acts as a "password" and the strings $s^1, \ldots, s^n$ are the seeds to the pseudorandom functions. Each concept $c \in C_m$ takes m-bit strings as inputs; we view such an m-bit input as a 4-tuple (b, x, i, j), where $b \in \{0,1\}$, $x \in \{0,1\}^n$, and $i, j \in \{0,1\}^{\log n}$ each represent a number in the range $\{1, 2, \ldots, n\}$.

Let $\{h_s^0 : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ be a pseudorandom permutation family and let $\{h_s^1 : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ be the pseudorandom function family from Theorem 18, so that $h_s^1(x) = h_s^1(x \oplus s)$. The concept $c_{y,\tilde{s}}$ is defined as follows on input (b, x, i, j) :

- **If $b = 0$**: A query $(0, x, i, j)$ is called a *function query.* The value of $c_{y,\tilde{s}}(0, x, i, j)$ is $h_{s^i}^{y_i}(x)_j$, i.e., the jth bit of the n-bit string $h_{s^i}^{y_i}(x)$. Thus the bit y_i determines whether the ith pseudorandom function used is a permutation or is invariant under XOR mask with s^i.
- **If $b = 1$**: A query $(1, x, i, j)$ is called a *seed query.* The value of $c_{y,\tilde{s}}(1, x, i, j)$ is 0 if $x \neq y$ and is s_j^i (the jth bit of the ith seed s^i) if $x = y$.

The intuition behind our construction is simple: in order to learn the target concept successfully a learning algorithm must identify each seed string $s^1, \ldots, s^n$. These strings can be identified by making seed queries $(1, y, i, j)$, but in order to make the correct seed queries the learning algorithm must know y. Since each bit y_i corresponds to whether an oracle is a permutation or is XOR-mask invariant, a quantum algorithm can determine each y_i and thus can learn successfully. However, no p.p.t. algorithm can distinguish between these two types of oracles (since in either case the oracle is pseudorandom and hence is indistinguishable from a truly random function), and thus no p.p.t. algorithm can learn y.

8.2. A quantum algorithm which learns C in polynomial time. The main result of this section is the following.

THEOREM 19. *The concept class C described above is polynomial-time learnable from quantum membership queries.*

Proof. Let $c_{y,\tilde{s}} \in C_m$ be the target concept. Each function $h_{s^i}^{y_i}$ is a permutation iff $y_i = 0$, and is XOR-mask invariant iff $y_i = 1$. (This is why we do not allow $s^i = 0^n$ in the definition of the concept class.) Using quantum membership queries, a poly(n)-time quantum algorithm can run Simon's procedure n times, once for each function $h_{s^i}^{y_i}$, and thus determine each bit y_i with high probability. (One detail which arises here is that Simon's algorithm uses an oracle $\{0,1\}^n \to \{0,1\}^n$, whereas in our learning setting the oracle outputs one bit at a time. This is not a problem since it is possible to simulate any call to Simon's oracle by making n sequential calls,

bit by bit, to our oracle.) Given the string $y = y_1 \cdots y_n$, the algorithm can then make n^2 queries on inputs $(1, y, i, j)$ for $1 \leq i, j \leq n$ to learn each of the n strings $s^1, \ldots, s^n$. Once y and $s^1, \ldots, s^n$ are known, it is straightforward to output a circuit for $c_{y,\tilde{s}}$. □

8.3. No classical algorithm learns C in polynomial time. The main result of this section is the following.

THEOREM 20. *C is not polynomial-time learnable from classical membership queries.*

Let $C'_m \supset C_m$, $|C'_m| = 2^{n^2+n}$, be the concept class $C'_m = \{c_{y,\tilde{s}} : y, s^1, \ldots, s^n \in \{0,1\}^n\}$; thus C'_m includes concepts in which s^i may be 0^n. The following lemma states that it is hard to learn a target concept chosen uniformly from C'_m.

LEMMA 21. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n,*

$$\Pr_{c_{y,\tilde{s}} \in C'_m}[A^{c_{y,\tilde{s}}} \text{ outputs a hypothesis } h \equiv c_{y,\tilde{s}}] < \frac{1}{Q(n)}.$$

To see that Lemma 21 implies Theorem 20, we note that the uniform distribution over C'_m and the uniform distribution over C_m are nearly identical (the two distributions have total variation distance $O(n/2^n)$). Lemma 21 thus has the following analogue for C_m, which clearly implies Theorem 20.

LEMMA 22. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n,*

$$\Pr_{c_{y,\tilde{s}} \in C_m}[A^{c_{y,\tilde{s}}} \text{ outputs a hypothesis } h \equiv c_{y,\tilde{s}}] < \frac{1}{Q(n)}.$$

The proof of Lemma 21 proceeds as follows: we say that a learning algorithm A *hits* y if at some point during its execution A makes a seed query $(1, y, i, j)$, and we say that A *misses* y if A does not hit y. We have that

$$\begin{aligned}\Pr_{c_{y,\tilde{s}} \in C'_m}[A^{c_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}}] &= \Pr[A^{c_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}} \ \ \& \ \ A^{c_{y,\tilde{s}}} \text{ hits } y] \\ &\quad + \Pr[A^{c_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}} \ \ \& \ \ A^{c_{y,\tilde{s}}} \text{ misses } y] \\ &\leq \Pr[A^{c_{y,\tilde{s}}} \text{ hits } y] \\ &\quad + \Pr[A^{c_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}} \ \mid \ A^{c_{y,\tilde{s}}} \text{ misses } y].\end{aligned}$$

Lemma 21 thus follows from the following two lemmas.

LEMMA 23. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n,*

$$\Pr_{c_{y,\tilde{s}} \in C'_m}[A^{c_{y,\tilde{s}}} \text{ hits } y] < \frac{1}{Q(n)}.$$

LEMMA 24. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n,*

$$\Pr_{c_{y,\tilde{s}} \in C'_m}[A^{c_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}} \ \mid \ A^{c_{y,\tilde{s}}} \text{ misses } y] < \frac{1}{Q(n)}.$$

8.3.1. Proof of Lemma 23. The idea of the proof is as follows: before hitting y for the first time, algorithm A gets 0 as the answer to each seed query, so A might as well be querying a modified oracle which answers 0 to *every* seed query. We show that no p.p.t. algorithm which has access to such an oracle can output y with inverse polynomial success probability (intuitively this is because such an oracle consists entirely of pseudorandom functions and hence can provide no information to any p.p.t. algorithm), and thus A's probability of hitting y must be less than $1/\mathrm{poly}(n)$ as well.

More formally, let A be any p.p.t. learning algorithm. Without loss of generality we may suppose that A always makes exactly $q(n)$ seed queries during its execution for some polynomial q. Let $X^1, \ldots, X^{q(n)}$ be the sequence of strings in $\{0,1\}^n$ on which $A^{c_{y,\tilde{s}}}$ makes its seed queries; i.e., $A^{c_{y,\tilde{s}}}$ uses $(1, X^t, i_t, j_t)$ as its tth seed query. Each X^t is a random variable over the probability space defined by the uniform choice of $c_{y,\tilde{s}} \in C'_m$ and any internal randomness of algorithm A.

For each $c_{y,\tilde{s}} \in C'_m$ let $\tilde{c}_{y,\tilde{s}} : \{0,1\}^m \to \{0,1\}$ be a modified version of $c_{y,\tilde{s}}$ which answers 0 to all seed queries; i.e., $\tilde{c}_{y,\tilde{s}}(b,x,i,j)$ is $c_{y,\tilde{s}}(b,x,i,j)$ if $b = 0$ and is 0 if $b = 1$. Consider the following algorithm B which takes access to an oracle for $\tilde{c}_{y,\tilde{s}}$ and outputs an n-bit string. B executes algorithm $A^{\tilde{c}_{y,\tilde{s}}}$ (note that the oracle used is $\tilde{c}_{y,\tilde{s}}$ rather than $c_{y,\tilde{s}}$), then chooses a uniform random value $1 \le t \le q(n)$, and outputs $\tilde{X}^t$, the string on which $A^{\tilde{c}_{y,\tilde{s}}}$ made its tth seed query. Like the X^t's, each $\tilde{X}^t$ is a random variable over the probability space defined by a uniform choice of $c_{y,\tilde{s}} \in C'_m$ and any internal randomness of A.

The following two lemmas together imply Lemma 23.

LEMMA 25. $2q(n)^2 \cdot \Pr_{c_{y,\tilde{s}} \in C'_m}[B^{\tilde{c}_{y,\tilde{s}}} \textit{ outputs } y] \ge \Pr_{c_{y,\tilde{s}} \in C'_m}[A^{c_{y,\tilde{s}}} \textit{ hits } y]$.

LEMMA 26. *For all polynomials Q and all sufficiently large n, we have*

$$\left| \Pr_{c_{y,\tilde{s}} \in C'_m}[B^{\tilde{c}_{y,\tilde{s}}} \textit{ outputs } y] - \frac{1}{2^n} \right| < \frac{1}{Q(n)}.$$

Proof of Lemma 25. We have that

$$\sum_{t=1}^{q(n)} \Pr[X^t = y \text{ and } X^\tau \neq y \text{ for } \tau < t] \le \sum_{t=1}^{q(n)} \Pr[X^t = y \mid X^\tau \neq y \text{ for } \tau < t].$$

Since the left-hand side of this inequality is exactly $\Pr_{c_{y,\tilde{s}} \in C'_m}[A^{c_{y,\tilde{s}}} \text{ hits } y]$, for some value $1 \le t_0 \le q(n)$ we have

$$\Pr[X^{t_0} = y \mid X^\tau \neq y \text{ for } \tau < t_0] \ge \Pr[A^{c_{y,\tilde{s}}} \text{ hits } y]/q(n). \tag{2}$$

Since the distribution of responses to function queries that A makes prior to its first seed query is the same regardless of whether the oracle is $c_{y,\tilde{s}}$ or $\tilde{c}_{y,\tilde{s}}$, it is clear that the random variables X^1 and $\tilde{X}^1$ are identically distributed. An inductive argument shows that for all $t \ge 1$, the conditional random variables $X^t \mid (X^\tau \neq y \text{ for } \tau < t)$ and $\tilde{X}^t \mid (\tilde{X}^\tau \neq y \text{ for } \tau < t)$ are identically distributed. (In each case the conditioning ensures that the distribution of responses to seed queries that A makes prior to its tth seed query is the same, i.e., all 0.)

We consider two possible cases. If $\Pr_{c_{y,\bar{s}} \in C'_m}[\tilde{X}^\tau \neq y \text{ for } \tau < t_0] > 1/2$, then

$$\begin{aligned}
\Pr_{c_{y,\bar{s}} \in C'_m}[B^{\tilde{c}_{y,\bar{s}}} \text{ outputs } y] &\geq \Pr[B^{\tilde{c}_{y,\bar{s}}} \text{ chooses } t_0] \cdot \Pr[\tilde{X}^{t_0} = y \ \&\ \tilde{X}^\tau \neq y \text{ for } \tau < t_0] \\
&= \frac{\Pr[\tilde{X}^{t_0} = y \mid \tilde{X}^\tau \neq y \text{ for } \tau < t_0] \cdot \Pr[\tilde{X}^\tau \neq y \text{ for } \tau < t_0]}{q(n)} \\
&> \frac{\Pr[\tilde{X}^{t_0} = y \mid \tilde{X}^\tau \neq y \text{ for } \tau < t_0]}{2q(n)} \\
&= \frac{\Pr[X^{t_0} = y \mid X^\tau \neq y \text{ for } \tau < t_0]}{2q(n)} \\
&\geq \frac{\Pr[A^{c_{y,\bar{s}}} \text{ hits } y]}{2q(n)^2}. \qquad \text{(by (2))}
\end{aligned}$$

Otherwise if $\Pr_{c_{y,\bar{s}} \in C'_m}[\tilde{X}^\tau \neq y \text{ for } \ \tau < t_0] \leq 1/2$, then $\sum_{t=1}^{t_0-1} \Pr[\tilde{X}^t = y] \geq 1/2$ and hence $\Pr_{c_{y,\bar{s}} \in C'_m}[B^{\tilde{c}_{y,\bar{s}}} \text{ outputs } y]$ is at least

$$\sum_{t=1}^{t_0-1} \Pr[B^{\tilde{c}_{y,\bar{s}}} \text{ chooses } t] \cdot \Pr[\tilde{X}^t = y] \geq \frac{1}{2q(n)} \geq \frac{\Pr[A^{c_{y,\bar{s}}} \text{ hits } y]}{2q(n)^2}. \qquad \square$$

Proof of Lemma 26. For $z, \zeta \in \{0,1\}^n$ let

$$p_\zeta^z = \Pr_{c_{y,\bar{s}} \in C'_m}[B^{\tilde{c}_{y,\bar{s}}} \text{ outputs } z \ \mid \ y = \zeta].$$

For $\ell \in \{1, \ldots, n\}$ let $\zeta || \ell$ denote ζ with the ℓth bit flipped. Similarly, for $S \subseteq \{1, \ldots, n\}$ let $\zeta || S$ denote ζ with bits flipped in all positions corresponding to S.

Fix $z, \zeta \in \{0,1\}^n$ and $\ell \in \{1, \ldots, n\}$ and consider the following algorithm $D_{z,\zeta,\ell}$, which takes access to an oracle $f : \{0,1\}^n \to \{0,1\}^n$ and outputs a single bit. For all $i \neq \ell$, algorithm $D_{z,\zeta,\ell}$ first chooses a random n-bit string s^i. $D_{z,\zeta,\ell}$ then runs algorithm B, simulating the oracle for B as follows:

- queries $(0, x, \ell, j)$ are answered with the bit $f(x)_j$,
- for $i \neq \ell$ queries $(0, x, i, j)$ are answered with the bit $h_{s^i}^{\zeta_i}(x)_j$,
- all queries $(1, x, i, j)$ are answered with the bit 0.

Finally algorithm $D_{z,\zeta,\ell}$ outputs 1 if B's output is z and outputs 0 otherwise.

It is easy to verify that for all z, ζ, ℓ we have

$$p_\zeta^z = \Pr_{s \in \{0,1\}^n}[D_{z,\zeta,\ell}^{h_s^{\zeta_\ell}} \text{ outputs } 1]$$

and

$$p_{\zeta||\ell}^z = \Pr_{s \in \{0,1\}^n}[D_{z,\zeta,\ell}^{h_s^{1-\zeta_\ell}} \text{ outputs } 1].$$

From the definition of pseudorandomness and the triangle inequality it follows that $|p_\zeta^z - p_{\zeta||\ell}^z| < \frac{1}{nQ(n)}$. Making $|S| \leq n$ applications of this inequality and using the triangle inequality, we find that

$$|p_\zeta^z - p_{\zeta||S}^z| < \frac{1}{Q(n)}.$$

We thus have that $|p_\zeta^z - p_z^z| < \frac{1}{Q(n)}$ for all $z, \zeta \in \{0,1\}^n$. Since $\sum_{z\in\{0,1\}^n} p_\zeta^z = 1$, we have that

$$\begin{aligned}\left|\Pr_{c_{y,\tilde{s}}\in C'_m}[B^{\tilde{c}_{y,\tilde{s}}} \text{ outputs } y] - \frac{1}{2^n}\right| &= \left|\frac{1}{2^n}\left(\sum_{z\in\{0,1\}^n} p_z^z\right) - \frac{1}{2^n}\right| \\ &= \frac{1}{2^n}\left|\sum_{z\in\{0,1\}^n}(p_z^z - p_\zeta^z)\right| \\ &< \frac{1}{Q(n)}. \qquad \square\end{aligned}$$

8.3.2. Proof of Lemma 24. The idea here is that conditioning on the event that A misses y ensures that the only information which A has about y and $\tilde{s}$ comes from querying oracles for the pseudorandom functions $h_{s^i}^{y_i}$. Since these pseudorandom functions are indistinguishable from truly random functions, no p.p.t. algorithm can learn successfully.

Formally, let A be any p.p.t. learning algorithm. Consider the following algorithm B which takes access to an oracle $h_{s^n}^{y_n} : \{0,1\}^n \to \{0,1\}^n$ and outputs a representation of a function $g : \{0,1\}^n \to \{0,1\}^n$. Algorithm B first chooses $\hat{y} = y_1 \cdots y_{n-1}$ uniformly from $\{0,1\}^{n-1}$ and chooses $n-1$ strings $s^1, \ldots, s^{n-1}$ each uniformly from $\{0,1\}^n$. B then runs algorithm $A^{\tilde{c}_{y,\tilde{s}}}$ (observe that B can simulate the oracle $\tilde{c}_{y,\tilde{s}}$ since it has access to an oracle for $h_{s^n}^{y_n}$ and knows y_i, s^i for $i \neq n$), which generates some hypothesis h. Finally B outputs the function $g : \{0,1\}^n \to \{0,1\}^n$ defined by $g(x) \stackrel{\text{def}}{=} h(0,x,n,1)h(0,x,n,2)\cdots h(0,x,n,n)$.

The following two lemmas together imply Lemma 24.

LEMMA 27. *For all sufficiently large n,*

$$\begin{aligned}&\Pr_{y_n\in\{0,1\}, s^n\in\{0,1\}^n}[B^{h_{s^n}^{y_n}} \textit{ outputs } g \equiv h_{s^n}^{y_n}] \\ &\qquad > \frac{\Pr_{c_{y,\tilde{s}}\in C'_m}[A^{c_{y,\tilde{s}}} \textit{ outputs } h \equiv c_{y,\tilde{s}} \mid A^{c_{y,\tilde{s}}} \textit{ misses } y]}{2}.\end{aligned}$$

LEMMA 28. *For all polynomials Q and all sufficiently large n, we have*

$$\Pr_{y_n\in\{0,1\}, s^n\in\{0,1\}^n}[B^{h_{s^n}^{y_n}} \textit{ outputs } g \equiv h_{s^n}^{y_n}] < \frac{1}{Q(n)}.$$

Proof of Lemma 27. It is easy to see that if $A^{\tilde{c}_{y,\tilde{s}}}$ outputs a hypothesis which is equivalent to $c_{y,\tilde{s}}$, then g will be equivalent to $h_{s^n}^{y_n}$. For sufficiently large n we thus have that $\Pr_{y_n\in\{0,1\}, s^n\in\{0,1\}^n}[B^{h_{s^n}^{y_n}} \text{ outputs } g \equiv h_{s^n}^{y_n}]$ is at least

$$\begin{aligned}\Pr_{c_{y,\tilde{s}}\in C'_m}[A^{\tilde{c}_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}}] &\geq \Pr[A^{\tilde{c}_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}} \text{ and } A^{\tilde{c}_{y,\tilde{s}}} \text{ misses } y] \\ &= \Pr[A^{\tilde{c}_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}} \mid A^{\tilde{c}_{y,\tilde{s}}} \text{ misses } y] \\ &\quad \cdot \Pr[A^{\tilde{c}_{y,\tilde{s}}} \text{ misses } y] \\ &> \frac{\Pr[A^{\tilde{c}_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}} \mid A^{\tilde{c}_{y,\tilde{s}}} \text{ misses } y]}{2},\end{aligned}$$

where the last inequality follows from Lemma 23.

Let $TRANS(A^{c_{y,\tilde{s}}})$ ($TRANS(A^{\tilde{c}_{y,\tilde{s}}})$, respectively) denote a complete transcript of algorithm A's execution on oracle $c_{y,\tilde{s}}$ ($\tilde{c}_{y,\tilde{s}}$, respectively). $TRANS(A^{c_{y,\tilde{s}}})$ and $TRANS(A^{\tilde{c}_{y,\tilde{s}}})$ are each random variables over the probability space defined by a uniform choice of $c_{y,\tilde{s}} \in C'_m$ and any internal randomness of algorithm A. An easy induction shows that the two conditional random variables $TRANS(A^{\tilde{c}_{y,\tilde{s}}}) \mid (A^{\tilde{c}_{y,\tilde{s}}}$ misses $y)$ and $TRANS(A^{c_{y,\tilde{s}}}) \mid (A^{c_{y,\tilde{s}}}$ misses $y)$ are identically distributed. This implies that

$$\Pr_{c_{y,\tilde{s}} \in C'_m}[A^{\tilde{c}_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}} | A^{\tilde{c}_{y,\tilde{s}}} \text{ misses } y] = \Pr_{c_{y,\tilde{s}} \in C'_m}[A^{c_{y,\tilde{s}}} \text{ outputs } h \equiv c_{y,\tilde{s}} | A^{c_{y,\tilde{s}}} \text{ misses } y],$$

which combined with the inequality above proves the lemma. □

Proof of Lemma 28. The following fact, which follows easily from the pseudorandomness of $\{h^0\}$ and $\{h^1\}$, states that $\{h^b_s\}_{b \in \{0,1\}, s \in \{0,1\}^n}$ is a pseudorandom function family.

FACT 4. *For all polynomials Q, p.p.t. oracle algorithms A, and sufficiently large n, we have*

$$\left| \Pr_{b \in \{0,1\}, s \in \{0,1\}^n}[A^{h^b_s} \textit{ outputs } 1] - \Pr_{F \in \mathcal{F}_n}[A^F \textit{ outputs } 1] \right| < \frac{1}{Q(n)}.$$

Intuitively the pseudorandomness of $\{h^b_s\}$ should make it hard for $B^{h^{y_n}_{s^n}}$ to output $h^{y_n}_{s^n}$ since clearly no p.p.t. algorithm, given oracle access to a truly random function F, could output a function equivalent to F. Formally, we consider an algorithm D which takes oracle access to a function $f : \{0,1\}^n \to \{0,1\}^n$ and outputs a single bit. D runs B^f to obtain a function g and then selects a string $z \in \{0,1\}^n$ which was not used as an oracle query in the computation of B^f. D calls the oracle to obtain $f(z)$, evaluates g to obtain $g(z)$, and outputs 1 if the two values are equal and 0 otherwise.

Clearly $\Pr[D^f \text{ outputs } 1] \geq \Pr[B^f \text{ outputs } g \equiv f]$. Since $\Pr_{F \in \mathcal{F}_n}[D^F \text{ outputs } 1] = 1/2^n$, using Fact 4, we find that

$$\left| \Pr_{y_n \in \{0,1\}, s^n \in \{0,1\}^n}[D^{h^{y_n}_{s^n}} \text{ outputs } 1] - \frac{1}{2^n} \right| < \frac{1}{2Q(n)}$$

and hence

$$\Pr_{y_n \in \{0,1\}, s^n \in \{0,1\}^n}[B^{h^{y_n}_{s^n}} \text{ outputs } g \equiv h^{y_n}_{s^n}] < \frac{1}{Q(n)}. \quad \square$$

9. Breaking classical cryptography in a quantum setting. Our constructions highlight some interesting issues concerning the relation between quantum oracle computation and classical cryptography. It is clear that a quantum algorithm, given access to a quantum black-box oracle for an unknown function, can efficiently distinguish between truly random functions and pseudorandom functions drawn from the family $\{g_s\}$ of Theorem 18. Our construction of $\{g_s\}$ thus shows that cryptographic constructions which are provably secure in the classical model can fail in a quantum setting. We emphasize that this failure does *not* depend on the ability of polynomial-time quantum algorithms to invert particular one-way functions such as factoring; even if no quantum algorithm can efficiently invert the one-way function used to construct $\{g_s\}$, our results show that a polynomial-time quantum algorithm can be a successful distinguisher. It would be interesting to obtain stronger constructions of pseudorandom functions which are provably secure in the quantum oracle framework.

10. Conclusion and future directions. While we have shown that quantum and classical learning are information-theoretically equivalent up to polynomial factors, we have not attempted to obtain the tightest possible bounds relating the two query complexities. In another direction, while we have shown the existence of concept classes which separate efficient quantum and classical learning, many questions remain about the relationship between efficient quantum and classical learnability for natural concept classes studied in learning theory. It would be interesting to develop efficient quantum learning algorithms for natural concept classes, such as the polynomial-time quantum algorithm of Bshouty and Jackson [13] for learning DNF formulae from uniform quantum examples.

Appendix: Proof of Theorem 18. We say that a polynomial-time deterministic algorithm $G : \{0,1\}^n \to \{0,1\}^{2n}$ is a *pseudorandom generator* if for all polynomials Q, all p.p.t. algorithms A, and all sufficiently large n,

$$\left| \Pr_{z \in \{0,1\}^n} [A(G(z)) \text{ outputs } 1] - \Pr_{z \in \{0,1\}^{2n}} [A(z) \text{ outputs } 1] \right| < \frac{1}{Q(n)}.$$

Thus a pseudorandom generator is an efficient algorithm which converts an n-bit random string into a $2n$-bit string which "looks random" to any polynomial-time algorithm. Håstad et al. [25] have shown that pseudorandom generators exist if any one-way function exists.

For G a pseudorandom generator and $s \in \{0,1\}^n$ we write $G_0(s)$ to denote the first n bits of $G(s)$, and $G_1(s)$ to denote the last n bits of $G(s)$. For $x, s \in \{0,1\}^n$ let $f_s : \{0,1\}^n \to \{0,1\}^n$ be defined as

$$f_s(x) \stackrel{\text{def}}{=} G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_2}(G_{x_1}(s)))\cdots)).$$

In [23] it is shown that $\{f_s\}$ is a pseudorandom function family. We now show that the family $\{g_s\}$ defined by $g_s(x) \stackrel{\text{def}}{=} f_s(x) \oplus f_s(x \oplus s)$ is pseudorandom.

Let $\mathcal{F}'_n$ be the following probability distribution over functions from $\{0,1\}^n$ to $\{0,1\}^n$: a function F' is drawn from $\mathcal{F}'_n$ by drawing a random function F from $\mathcal{F}_n$, drawing a random string $s \in \{0,1\}^n$, and letting F' be the function defined as $F'(x) = F(x) \oplus F(x \oplus s)$. Theorem 18 follows from the following two lemmas.

LEMMA 29. *For all polynomials Q, all p.p.t. oracle algorithms M, and all sufficiently large n,*

$$\left| \Pr_{F \in \mathcal{F}_n} [M^F \text{ outputs } 1] - \Pr_{F' \in \mathcal{F}'_n} [M^{F'} \text{ outputs } 1] \right| < \frac{1}{Q(n)}.$$

Proof. Consider an execution of M with an oracle $F' \in \mathcal{F}'_n$ defined by $F'(x) = F(x) \oplus F(x \oplus s)$. Let $S = \{x^1, \ldots, x^t\} \subset \{0,1\}^n$ be the set of strings which M uses as queries to F'. We say that M *finds* s if $x^i = x^j \oplus s$ for some $x^i, x^j \in S$. If M does not find s, then the distribution of answers which M receives from F' is identical to the distribution which M would receive if it were querying a random function $F \in \mathcal{F}_n$, since in both cases each distinct query is answered with a uniformly distributed n-bit string. Thus the left side of the inequality above is at most $\Pr[M \text{ finds } s]$. A simple inductive argument given in the proof of Theorem 3.3 of [40] shows that this probability is at most $\sum_{k=1}^{t} (k/(2^n - (k-2)(k-1)/2))$. Since M is polynomial-time, t is at most $\text{poly}(n)$, and the lemma follows. ∎

LEMMA 30. *For all polynomials Q, all p.p.t. oracle algorithms A, and all sufficiently large n,*

$$\left| \Pr_{F' \in \mathcal{F}'_n}[M^{F'} \text{ outputs } 1] - \Pr_{s \in \{0,1\}^n}[M^{g_s} \text{ outputs } 1] \right| < \frac{1}{Q(n)}.$$

Proof. We require the following fact which is due to Yao [44].

FACT 5. *Let G be a pseudorandom generator, let $q(n)$ and $Q(n)$ be polynomials, and let M_* be a p.p.t. algorithm which takes as input $q(n)$ strings each of length $2n$ bits. Then for all sufficiently large n we have*

$$|p_n^G - p_n^U| < \frac{1}{Q(n)},$$

where p_n^U is the probability that M_* outputs 1 on input $q(n)$ random strings in $\{0,1\}^{2n}$ and p_n^G is the probability that M_* outputs 1 on input $q(n)$ strings each of which is obtained by applying G to a random string from $\{0,1\}^n$.

We prove Lemma 30 by contradiction; suppose that there exists a p.p.t. oracle algorithm M and a polynomial Q such that for infinitely many values of n,

$$\left| \Pr_{F' \in \mathcal{F}'_n}[M^{F'} \text{ outputs } 1] - \Pr_{s \in \{0,1\}^n}[M^{g_s} \text{ outputs } 1] \right| \geq \frac{1}{Q(n)}. \tag{3}$$

We will show that there is a p.p.t. algorithm M_* which contradicts Fact 5.

As in the proof in [23] that $\{f_s\}$ is a pseudorandom function family, we use a so-called "hybrid" argument. Consider the following algorithms A_i ($i = 0, 1, \ldots, n$), each of which defines a mapping from $\{0,1\}^n$ to $\{0,1\}^n$ and hence could conceivably be used as an oracle to answer M's queries. Conceptually, each algorithm A_i contains a full binary tree of depth n in which the root (at depth 0) is labeled with a random n-bit string s; if $i > 0$, then each node at depth i is also labeled with an independently chosen random n-bit string. Each node at depth $j > i$ also has an n-bit label determined as follows: if node v has label z, then the left child of v has label $G_0(z)$, and the right child of v has label $G_1(z)$. Each node in the tree has an *address* which is a binary string: the root's address is the empty string, and if a node has address $\alpha \in \{0,1\}^*$, then its left child has address $\alpha 0$ and its right child has address $\alpha 1$ (so each leaf has a different n-bit string as its address). Let $L(x)$ denote the label of the node whose address is x. Algorithm A_i answers a query $x \in \{0,1\}^n$ with the n-bit string $L(x) \oplus L(x \oplus s)$.

(Note that algorithm A_i need not precompute any leaf labels. Instead, A_i can run in poly(n) time at each invocation by randomly choosing s once-and-for-all the first time it is invoked and labeling the necessary portion of the tree "on the fly" at each invocation by choosing random strings for the depth-i nodes as required and computing descendants' labels as described above. A_i must store the random strings which it uses to label depth-i nodes so as to maintain consistency over successive invocations.)

For $i = 0, 1, \ldots, n$ let p_n^i denote $\Pr[M^{A_i} \text{ outputs } 1]$, i.e., the probability that M outputs 1 if its oracle queries are answered by algorithm A_i. Let p_n^g denote $\Pr_{s \in \{0,1\}^n}[M^{g_s} \text{ outputs } 1]$ and $p_n^{F'}$ denote $\Pr_{F' \in \mathcal{F}'_n}[M^{F'} \text{ outputs } 1]$. We have that $p_n^0 = p_n^g$ since algorithm A_0 behaves exactly like an oracle for g_s, where s is a random n-bit string. We also have that $p_n^n = p_n^{F'}$ since algorithm A_n behaves exactly like an oracle for $F' \in \mathcal{F}'_n$. Inequality (3) thus implies that $|p_n^0 - p_n^n| \geq 1/Q(n)$ for infinitely many values of n.

Now we describe the algorithm M_*, which distinguishes between sets of strings. Let $q(n)$ be a polynomial which bounds the running time of M on inputs of length n (so M makes at most $q(n)$ oracle queries, given access to an oracle from $\{0,1\}^n$ to $\{0,1\}^n$). The algorithm M_* takes as input a set U_n of $2q(n)$ strings of length $2n$. M_* works by first selecting a uniform random value $0 \le i \le n-1$ and a uniform random string $s \in \{0,1\}^n$. M_* then runs algorithm M, answering M's oracle queries as follows. There are two cases, depending on whether or not the prefix $s_1 \dots s_i$ is 0^i:

- *Case* 1. $s_1 \dots s_i \neq 0^i$. Let $x = x_1 \dots x_n$ be the query string. If no earlier query string had prefix $x_1 \dots x_i$ or $(x_1 \oplus s_1) \dots (x_i \oplus s_i)$, then M_* takes the next two $2n$-bit strings from U_n; call these strings $u^1 = u^1_0 u^1_1$ and $u^2 = u^2_0 u^2_1$, where $|u^i_j| = n$. Now M_* stores the four pairs

$$(x_1 \dots x_i 0, u^1_0),\ (x_1 \dots x_i 1, u^1_1),\ ((x_1 \oplus s_1) \dots (x_i \oplus s_i)0, u^2_0),\ \text{and}$$
$$((x_1 \oplus s_1) \dots (x_i \oplus s_i)1, u^2_1)$$

  and answers with the string

$$(*) \qquad G_{x_n}(G_{x_{n-1}}(\dots G_{x_{i+2}}(u^1_{x_{i+1}}) \dots)) \oplus G_{x_n \oplus s_n}(G_{x_{n-1} \oplus s_{n-1}}(\dots G_{x_{i+2} \oplus s_{i+2}}(u^2_{x_{i+1} \oplus s_{i+1}}) \dots)).$$

  Otherwise, if an earlier query string had prefix $x_1 \dots x_i$ or $(x_1 \oplus s_1) \dots (x_i \oplus s_i)$, then instead M_* retrieves the two previously stored pairs

$$(x_1 \dots x_i x_{i+1}, u^1_{x_{i+1}}) \quad \text{and} \quad ((x_1 \oplus s_1) \dots (x_i \oplus s_i)(x_{i+1} \oplus s_{i+1}), u^2_{x_{i+1} \oplus s_{i+1}})$$

  and answers with $(*)$ as above.
- *Case* 2. $s_1 \dots s_i = 0^i$. Let $x = x_1 \dots x_n$ be the query string. If no earlier query string had prefix $x_1 \dots x_i$, then M_* takes the next $2n$-bit string from U_n; call this string $u = u_0 u_1$, where $|u_0| = |u_1| = n$. Now M_* stores the two pairs

$$(x_1 \dots x_i 0, u_0),\ (x_1 \dots x_i 1, u_1)$$

  and answers with

$$(**) \qquad G_{x_n}(G_{x_{n-1}}(\dots G_{x_{i+2}}(u_{x_{i+1}}) \dots)) \oplus G_{x_n \oplus s_n}(G_{x_{n-1} \oplus s_{n-1}}(\dots G_{x_{i+2} \oplus s_{i+2}}(u_{x_{i+1} \oplus s_{i+1}}) \dots)).$$

  Otherwise, if an earlier query string had prefix $x_1 \dots x_i$, then M_* retrieves the two pairs

$$(x_1 \dots x_i x_{i+1}, u_{x_{i+1}}) \qquad \text{and} \qquad (x_1 \dots x_i (x_{i+1} \oplus s_{i+1}), u_{x_{i+1} \oplus s_{i+1}})$$

  (these two pairs are the same if $s_{i+1} = 0$) and answers with $(**)$ as above.

The crucial properties of algorithm M_*, which are straightforwardly verified, are the following: if each string in U_n is generated by applying G to a random n-bit string (scenario 1), then M_* simulates a computation of M with oracle A_i. On the other hand, if each string in U_n is chosen uniformly from $\{0,1\}^{2n}$ (scenario 2), then M_* simulates a computation of M with oracle A_{i+1}.

It is easy to see now that in scenario 1 we have $\Pr[M_* \text{ outputs } 1] = \sum_{i=0}^{n-1} p^i_n / n$, while in scenario 2 we have $\Pr[M_* \text{ outputs } 1] = \sum_{i=1}^{n} p^{i+1}_n / n$. These two probabilities differ by $(1/n) \cdot |p^0_n - p^n_n|$, which is at least $1/nQ(n)$ for infinitely many values of n. Now by Fact 5, the existence of M_* contradicts the fact that G is a pseudorandom generator, and the lemma is proved. ∎

Acknowledgments. We thank R. de Wolf for helpful comments and suggestions, and A. Klivans for stimulating discussions.

REFERENCES

[1] A. Ambainis, *Quantum lower bounds by quantum arguments*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, Portland, OR, 2000, ACM, New York, pp. 636–643.
[2] D. Angluin, *Queries and concept learning*, Machine Learning, 2 (1988), pp. 319–342.
[3] D. Angluin and M. Kharitonov, *When won't membership queries help?*, J. Comput. Systems Sci., 50 (1995), pp. 336–355.
[4] M. Anthony and N. Biggs, *Computational Learning Theory: An Introduction*, Cambridge University Press, Cambridge, UK, 1997.
[5] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf, *Quantum lower bounds by polynomials*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, IEEE Computer Society, Piscataway, NJ, pp. 352–361.
[6] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and weaknesses of quantum computing*, SIAM J. Comput., 26 (1997), pp. 1510–1523.
[7] E. Bernstein and U. Vazirani, *Quantum complexity theory*, SIAM J. Comput., 26 (1997), pp. 1411–1473.
[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, *Learnability and the Vapnik–Chervonenkis dimension*, J. ACM, 36 (1989), pp. 929–965.
[9] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, *Tight bounds on quantum searching*, Fortschr. Phys., 46 (1998), pp. 493–505.
[10] G. Brassard, P. Høyer, and A. Tapp, *Quantum counting*, in Proceedings of the 25th EATCS International Conference on Automata, Languages and Programming, Aalborg, Denmark, 1998, Springer, New York, pp. 820–831.
[11] G. Brassard and P. Høyer, *An exact quantum polynomial-time algorithm for Simon's problem*, in the Fifth Israeli Symposium on Theory of Computers and Systems, Ramat Gan, Israel, 1997, IEEE Computer Society Press, Los Alamitos, CA, pp. 12–23.
[12] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon, *Oracles and queries that are sufficient for exact learning*, J. Comput. Syst. Sci., 52 (1996), pp. 421–433.
[13] N. H. Bshouty and J. C. Jackson, *Learning DNF over the uniform distribution using a quantum example oracle*, SIAM J. Comput., 28 (1999), pp. 1136–1153.
[14] H. Buhrman, R. Cleve, R. de Wolf, and C. Zalka, *Reducing error probability in quantum algorithms*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, New York, 1999, IEEE Computer Society, Piscataway, NJ, pp. 358–368.
[15] H. Buhrman, R. Cleve, and A. Wigderson, *Quantum vs. classical communication and computation*, in Proceedings of the 30th ACM Symposium on Theory of Computing, Dallas, TX, 1998, ACM, New York, 1998, pp. 63–68.
[16] R. Cleve, *An introduction to quantum complexity theory*, in Collected Papers on Quantum Computation and Quantum Information Theory, C. Macchiavello, G. M. Palma, and A. Zeilinger, eds., World Scientific, River Edge, NJ, 2000, pp. 103–127.
[17] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proc. Roy. Soc. London A, 439 (1992), pp. 553–558.
[18] R. de Wolf, *personal communication*, 2000.
[19] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, *How many functions can be distinguished with k quantum queries?*, Phys. Rev. A, 60 (1999), pp. 4331–4333.
[20] S. Fenner, L. Fortnow, S. Kurtz, and L. Li, *An oracle builder's toolkit*, in Proceedings of the Eighth Structure in Complexity Theory Conference, San Diego, CA, 1993, IEEE Computer Society Press, Los Alamitos, CA, pp. 120–131.
[21] L. Fortnow and J. Rogers, *Complexity limitations on quantum computation*, J. Comput. System. Sci., 59 (1999), pp. 240–252.
[22] R. Gavaldà, *The complexity of learning with queries*, in Proceedings of the Ninth Structure in Complexity Theory Conference, Amsterdam, 1994, IEEE Computer Society Press, Los Alamitos, CA, pp. 324–337.
[23] O. Goldreich, S. Goldwasser, and S. Micali, *How to construct random functions*, J. ACM, 33 (1986), pp. 792–807.
[24] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, Philadelphia, 1996, ACM, New York, pp. 212–219.
[25] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.

[26] T. Hegedüs, *Generalized teaching dimensions and the query complexity of learning*, in Proceedings of the Eighth Conference on Computer Learning Theory, Santa Cruz, CA, 1995, ACM Press, New York, pp. 108–117.
[27] L. Hellerstein, K. Pillaipakkamnatt, V. Raghavan, and D. Wilkins, *How many queries are needed to learn?*, J. ACM, 43 (1996), pp. 840–862.
[28] M. Kearns and L. Valiant, *Cryptographic limitations on learning Boolean formulae and finite automata*, J. ACM 41, (1994), pp. 67–95.
[29] M. Kearns and U. Vazirani, *An Introduction to Computational Learning Theory*, MIT Press, Cambridge, MA, 1994.
[30] M. Luby and C. Rackoff, *How to construct pseudorandom permutations from pseudrandom functions*, SIAM J. Comput., 17 (1988), pp. 373–386.
[31] M. Nielsen and I. Chuang, *Quantum Computation and Quantum information*, Cambridge University Press, Cambridge, UK, 2000.
[32] J. Ortega, *Matrix Theory: A Second Course*, Plenum Press, New York, 1987.
[33] J. Patarin, *How to construct pseudorandom and super pseudorandom permutations from one single pseudorandom function*, in Proceedings of Advances in Cryptography (EUROCRYPT '92), Workshop on the Theory and Application of Cryptographic Techniques, Balatonfured, Hungary, 1992, Springer, New York, pp. 256–266.
[34] J. Pierpzyk, *How to construct pseudorandom permutations from single pseudorandom functions*, in Proceedings of Advances in Cryptography (EUROCRYPT '90), Workshop on the Theory and Application of Cryptographic Techniques, Aarhus, Denmark, 1990, Springer, New York, pp. 140–150.
[35] J. Pierpzyk and B. Sadeghiyan, *A construction for super pseudorandom permutations from a single pseudorandom function*, in Proceedings of Advances in Cryptography (EUROCRYPT '92), Workshop on the Theory and Application of Cryptographic Techniques, Balatonfured, Hungary, 1992, Springer, New York, pp. 267–284.
[36] J. Preskill, *Lecture Notes on Quantum Computation*, 1998, available online at http://www.theory.caltech.edu/people/preskill/ph229/.
[37] R. Servedio and S. Gortler, *Quantum versus classical learnability*, in Proceedings of the 16th Conference on Computational Complexity, Chicago, 2001, IEEE Computer Society Press, Los Alamitos, CA, pp. 138–148.
[38] R. Servedio, *Separating quantum and classical learning*, in Proceedings of the 28th EATCS International Conference on Automata, Languages, and Programming, Heraklion, Crete, 2001, Springer, New York, pp. 1065–1080.
[39] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.
[40] D. R. Simon, *On the power of quantum computation*, SIAM J. Comput., 26 (1997), pp. 1474–1483.
[41] L. G. Valiant, *A theory of the learnable*, Comm. ACM, 27 (1984), pp. 1134–1142.
[42] J. H. Van Lint, *Introduction to Coding Theory*, Springer-Verlag, New York, 1992.
[43] V. N. Vapnik and A. Y. Chervonenkis, *On the uniform convergence of relative frequencies of events to their probabilities*, Theory Probab. Appl., 16 (1971), pp. 264–280.
[44] A. Yao, *Theory and applications of trapdoor functions*, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, Chicago, IL, 1982, pp. 80–91.
[45] A. C. Yao, *Quantum circuit complexity*, in Proceedings of the 34th IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 352–361.
[46] C. Zalka, *Grover's quantum searching algorithm is optimal*, Phys. Rev. A, 60 (1999), pp. 2746–2751.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1093–1108

ON THE HARDNESS OF GRAPH ISOMORPHISM*

JACOBO TORÁN†

Abstract. We show that the graph isomorphism problem is hard under DLOGTIME uniform AC^0 many-one reductions for the complexity classes NL, PL (probabilistic logarithmic space) for every logarithmic space modular class Mod_kL and for the class DET of problems NC^1 reducible to the determinant. These are the strongest known hardness results for the graph isomorphism problem and imply a randomized logarithmic space reduction from the perfect matching problem to graph isomorphism. We also investigate hardness results for the graph automorphism problem.

Key words. graph isomorphism, complexity, reducibility

AMS subject classifications. 05C60, 68Q17, 68Q15

DOI. 10.1137/S009753970241096X

1. Introduction.

The graph isomorphism problem (GI) consists in determining whether two given graphs are isomorphic—in other words, whether there is a bijection between the nodes of the graphs preserving the edges. This problem has been intensively studied, in part because of its many applications, and in part because it is one of the few problems in NP that has resisted all attempts to be classified as NP-complete, or within P. The best existing upper bound for the problem given by Luks and Zemlyachenko is $\exp\sqrt{cn\log n}$ (cf. [7]), but there is no evidence of this bound being optimal, and for many restricted graph classes, polynomial time algorithms are known. This is, for example, the case for planar graphs [19], graphs of bounded degree [29], or graphs with bounded eigenvalue multiplicity [6]. In some cases, like trees [28, 11] or graphs with colored vertices and bounded color classes [30], even NC algorithms for isomorphism are known.

Concerning the hardness of GI, there is evidence indicating that the problem is not NP-complete. On the one hand, the counting version of GI is known to be reducible to its decisional version [31], while for all known NP-complete problems the counting version seems to be much harder. On the other hand it has been shown that if GI were NP-complete, then the polynomial time hierarchy would collapse to its second level [9, 36]. Because of these facts, there is a common belief that GI does not contain enough structure or redundancy to be hard for NP. The question of whether GI is P-hard is also open, and, moreover, the known lower bounds in terms of hardness results for GI are surprisingly weak. It is only known that isomorphism for trees is hard for NC^1 and for L (logarithmic space) depending on the encoding of the input [23].

In this paper we improve the existing hardness results by showing that GI is hard for all complexity classes defined in terms of the number of accepting computations of a nondeterministic logarithmic space Turing machine.

The key ingredient in the proof of our results is a graph gadget showing that GI has enough structure to encode a modular addition gate. Using this fact, we are able to give for any $k \in \mathbb{N}$ an AC^0 many-one reduction from the circuit value problem for

*Received by the editors July 4, 2002; accepted for publication (in revised form) April 12, 2004; published electronically July 20, 2004. A preliminary version of this paper appeared in *Proceedings of the* 41*st Annual IEEE Symposium on Foundations of Computer Science*, 2000.
http://www.siam.org/journals/sicomp/33-5/41096.html
†Abteilung Theoretische Informatik, Universität Ulm, Oberer Eselsberg, 89069 Ulm, Germany (toran@informatik.uni-ulm.de).

addition mod k gates, which is complete for $\mathrm{Mod}_k\mathrm{L}$, to GI. $\mathrm{Mod}_k\mathrm{L}$ is the complexity class corresponding to sets recognized by nondeterministic logarithmic space machines in which the number of accepting computations satisfies a congruence modulo k [10], and it lies within NC^2. We show that a circuit with modular gates can be directly transformed into a graph in which any automorphism of a certain kind maps a special vertex encoding the output gate to a vertex encoding the output of the circuit. The graphs used in the reduction have degree 3 and its vertices can be partitioned into color classes of size k^2. Luks [30] has given an NC upper bound for the complexity of the isomorphism problem restricted to graphs with bounded color classes. For isomorphism in this class of graphs, the gap between our hardness results and the upper bound given by Luks is therefore small. In fact, in [27] we have recently shown that for graphs of bounded color classes of sizes 2 and 3, GI is complete for symmetric logarithmic space.

By a simple use of the Chinese remainder theorem, the hardness results for the modular classes can be transformed into hardness results for NL. It is interesting to observe that the graphs obtained in this reduction have automorphism groups in which the sizes of the orbits of some of the nodes depend on the input size, and therefore these graphs do not have classes of colored vertices of constant size, as in the modular case.

Using the recent result that division can be performed in TC^0 [15, 17, 18], and the fact that an NC^1 circuit can be encoded in an isomorphism problem [23], we can, moreover, prove that any logarithmic space counting function can be reduced to GI. In particular this implies that GI is many-one hard for $\mathrm{C}_=\mathrm{L}$ and for probabilistic logarithmic space, PL. The hardness results culminate in Theorem 4.9, where it is shown that GI is hard for DET, defined by Cook [13] as the class of problems NC^1 Turing reducible to the determinant.

The perfect matching problem is (as GI) another problem of the short list that has resisted classification in terms of completeness. It was shown in [5] that perfect matching is randomly (or nonuniformly) reducible to $\mathrm{Mod}_k\mathrm{L}$ for every k. From our results this implies a (random or nonuniform) reduction from matching to GI, which provides the first reduction between the two well-studied problems. Moreover, as a consequence of derandomization results from [21, 3, 25], under the natural hypothesis that there is a set in $\mathrm{DSPACE}(n)$ with circuits of size $2^{\Omega(n)}$, our reduction implies a many-one AC^0 (deterministic) reduction from perfect matching to GI.

The graph automorphism problem (GA), determining whether a given graph has a nontrivial automorphism, is known to be many-one reducible to GI and seems to be a slightly easier problem. We show in section 5 that the hardness results for GI hold also for GA.

2. Preliminaries. We assume familiarity with basic notions of complexity theory such as can be found in standard textbooks in the area. We will prove hardness results for several logarithmic space complexity classes: NL is the class of languages accepted by nondeterministic Turing machines using logarithmic space. The graph accessibility problem (GAP; given a directed graph with two designated nodes s and t, determine whether there is a path from s to t) is known to be complete for NL, even in the case of acyclic graphs with in-degree at most 2.

#L, defined in [4] analogously to Valiant's class #P, is the class of functions $f : \Sigma^* \to \mathbb{N}$ that count the number of accepting paths of a nondeterministic Turing machine M on input x. The computation of a #L function on an input x can be reduced to the problem computing the number paths from node s to node t in a

directed graph G_x. The complexity classes PL (probabilistic logarithmic space), $\mathrm{C_{=}L}$ (exact threshold in logarithmic space), and $\mathrm{Mod}_k\mathrm{L}$ (modular counting in logarithmic space, $k \geq 2$) can be defined in terms of #L functions:

$$\mathrm{PL} = \{A : \exists p \in \text{ Poly}, f \in \#\mathrm{L}, \ x \in A \Leftrightarrow f(x) \geq 2^{p(|x|)}\} \text{ [16, 35]}.$$

$$\mathrm{C_{=}L} = \{A : \exists p \in \text{ Poly}, f \in \#\mathrm{L}, \ x \in A \Leftrightarrow f(x) = 2^{p(|x|)}\} \text{ [2]}.$$

$$\mathrm{Mod}_k\mathrm{L} = \{A : \exists f \in \#\mathrm{L}, \ x \in A \Leftrightarrow f(x) = 1 \bmod k\} \text{ [10]}.$$

Mod_k circuits ($k \geq 2$) are circuits where the input variables (and the wires) can take values in $\mathbb{Z}_k$, and the gates compute addition in $\mathbb{Z}_k$. The evaluation problem for such circuits (given fixed values for the inputs, determine if the output value is, for example, 1) is complete for $\mathrm{Mod}_k\mathrm{L}$ under AC^0 many-one reductions. This is because a directed acyclic graph with in-degree at most 2, and two designated nodes, s, t, can be easily transformed into a mod_k circuit computing the residue of the number of paths from s to t in G, modulo k.

In some of the proofs we will make use of NC^1 circuits. These are families of logarithmic depth, polynomial size Boolean circuits of bounded fan-in over the basis $\{\wedge, \vee, \neg\}$. DET [13] is the class of problems NC^1 Turing reducible to the determinant, or, in other words, the class of problems that can be solved by NC^1 circuits with additional oracle gates that can compute the determinant of integer matrices.

The known relationships among the considered classes are as follows:

$$\mathrm{Mod}_k\mathrm{L} \subseteq \mathrm{DET},$$

$$\mathrm{NL} \subseteq \mathrm{C_{=}L} \subseteq \mathrm{PL} \subseteq \mathrm{DET}.$$

Looking at the known inclusions, the hardness of GI for DET implies hardness with respect to the other classes. We prove, however, the result for all the classes separately, showing how the graphs produced by the reductions increase in complexity.

2.1. Reducibilities. We prove our hardness results for the DLOGTIME uniform AC^0 many-one reducibility (in short AC^0 reducibility). A set A is AC^0 reducible to another set B if there is family of circuits $\{C_n \mid n \in \mathbb{N}\}$ where each circuit C_n contains only AND, OR, and NOT gates, has size $n^{O(1)}$ and depth $O(1)$, and for each x of length n, $x \in A \Leftrightarrow C_n(x) \in B$. Moreover, the uniformity condition requires that there be a DLOGTIME Turing machine with direct access to its input defining the circuit in the sense that the machine can recognize the direct connection language of C_n [34, 8]. This language consists of the set of tuples $\langle t, a, b, y\rangle$, where a and b are numbers of nodes in C_n, t is the type of a, b is a child of a, and y is a string of length n.

2.2. Graph isomorphism, automorphism, and promise isomorphism. An automorphism in an undirected graph $G = (V, E)$ is a permutation φ of the nodes that preserves adjacency. That is, for every $u, v \in V, (u, v) \in E \Leftrightarrow (\varphi(u), \varphi(v)) \in E$. An isomorphism between two graphs G, H is a bijection between their sets of vertices which preserves the edges. $G \simeq H$ denotes that G and H are isomorphic. GI is the problem

$$\mathrm{GI} = \{(G, H) \mid G \text{ and } H \text{ are isomorphic graphs}\},$$

and GA is defined as

$$\text{GA} = \{G |\ G \text{ has an automorphism different from the identity}\}.$$

A graph G in $\overline{\text{GA}}$ is called rigid. For technical reasons we will consider the set of graph pairs $((G,H),(I,J))$ with exactly one of the pairs consisting of isomorphic graphs:

$$\text{PGI} = \{((G,H),(I,J)) |\ G \simeq H \text{ if and only if } I \not\simeq J\}.$$

For a tuple in PGI we are given the promise of G being isomorphic to H or I being isomorphic to J, and the problem is to determine which one is the isomorphic pair.

Sometimes we will deal with graphs with colored vertices. A coloring with k colors is a function $f : V \to \{1,\ldots,k\}$. In an isomorphism between colored graphs, the colors have to be preserved. The isomorphism problem for colored graphs can be easily reduced (by AC^0 reductions) to graph isomorphism without colors (see, e.g., [26]).

In some cases we will consider the following restricted automorphism problem: Given a graph $G = (V,E)$ and two lists of nodes $(x_1,\ldots,x_k),(y_1,\ldots,y_k)$, is there an automorphism in G mapping x_i to y_i for $1 \le i \le k$? This problem is also easily reducible to GI. In order to check whether there is an automorphism with the desired properties one can make two copies of G, G' and G''. In G' each of the nodes x_i has color i and in G'' node y_i receives this color. All the other nodes are colored with a new color 0, for example. G' and G'' are isomorphic if and only if G has an automorphism with the mentioned properties.

3. Hardness for the modular counting classes. We show now that GI is hard for all the logarithmic space modular counting classes Mod_kL ($k \ge 2$). The idea for this proof is to simulate a modular gate with a graph gadget and then combine the gadgets for the different gates into a graph, whose automorphisms simulate the behavior of the modular circuit.

The gadgets are defined by the following graphs (shown in Figure 3.1 for the case $k = 2$).

DEFINITION 3.1. *Let $k \ge 2$, and denote by $\oplus$ the addition in $\mathbb{Z}_k$. We define the undirected graph $G^k = (V,E)$, given by the set of $k^2 + 3k$ nodes*

$$\begin{aligned} V = &\{x_a, y_a, z_a \mid a \in \{0,\ldots,k-1\} \\ &\cup \{u_{a,b} \mid a,b \in \{0,\ldots,k-1,\}\} \end{aligned}$$

and edges

$$\begin{aligned} E = &\{(x_a, u_{a,b}) \mid a,b \in \{0,\ldots,k-1\}\} \\ &\cup \{(y_b, u_{a,b}) \mid a,b \in \{0,\ldots,k-1\}\} \\ &\cup \{(u_{a,b}, z_{a\oplus b}) \mid a,b \in \{0,\ldots,k-1\}\}. \end{aligned}$$

The graph gadget for a modular gate has nodes encoding the inputs and outputs of the gate. Any automorphism in the graph mapping the input nodes in a certain way must map the output nodes according to the value of the modular gate being simulated.

LEMMA 3.2. *Fix $k \ge 2$; for any $a,b \in \{0,\ldots,k-1\}$,*

(1) *there is a unique automorphism φ in G^k mapping x_i to $x_{a\oplus i}$ and y_i to $y_{b\oplus i}$ for $i = 0,\ldots,k-1$; and*
(2) *this automorphism maps z_i to $z_{a\oplus b\oplus i}$.*

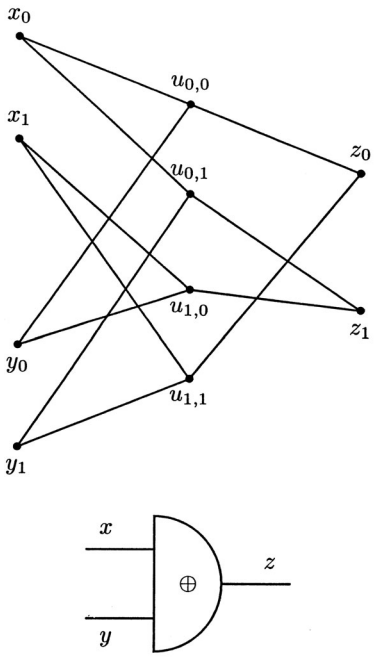


FIG. 3.1. *The graph G^2 simulating a parity gate.*

Proof. Let $a, b \in \{0, \ldots, k-1\}$, and denote by $\oplus$ the addition in $\mathbb{Z}_k$. We consider the following function $\varphi : V \to V$ defined as

$$\begin{aligned}
\varphi(x_i) &= x_{a\oplus i} \text{ for } i \in 0, \ldots, k-1, \\
\varphi(y_i) &= y_{b\oplus i} \text{ for } i \in 0, \ldots, k-1, \\
\varphi(u_{i,j}) &= u_{a\oplus i, b\oplus j} \text{ for } i, j \in 0, \ldots, k-1, \\
\varphi(z_i) &= z_{a\oplus b\oplus i} \text{ for } i \in 0, \ldots, k-1.
\end{aligned}$$

We prove first that φ is an automorphism. For this we have to show that for every pair of nodes v, w, $(v, w) \in E$ if and only if $(\varphi(v), \varphi(w)) \in E$. The nodes in graph G^k can be partitioned in three layers, the x and y nodes (input layer), the u nodes, and the z nodes (output layer). Edges exist only between nodes from the first and second layers, or between nodes from the second and third layers. We consider first an edge between the first two layers. Let $v = x_i$ and $w = u_{l,m}$ with $i, l, m \in \{0, \ldots, k-1\}$. Then $\varphi(v) = x_{a\oplus i}$ and $\varphi(w) = u_{a\oplus l, b\oplus m}$. By the definition of G^k,

$$\begin{aligned}
(x_i, u_{l,m}) \in E &\Leftrightarrow i = l \\
&\Leftrightarrow a \oplus i = a \oplus l \\
&\Leftrightarrow (x_{a\oplus i}, u_{a\oplus l, b\oplus m}) \in E \\
&\Leftrightarrow (\varphi(x_i), \varphi(u_{l,m})) \in E.
\end{aligned}$$

In the case $v = y_j$ the proof is analogous. For an edge (v, w) between the second and third layers, let $(v, w) = (u_{i,j}, z_l)$ with $i, j, l \in \{0, \ldots, k-1\}$. Then $\varphi(v) = u_{a\oplus i, b\oplus j}$

and $\varphi(w) = z_{a\oplus b\oplus l}$. By the definition of G^k,

$$\begin{aligned}(u_{i,j}, z_l) \in E &\Leftrightarrow i \oplus j = l \\ &\Leftrightarrow a \oplus b \oplus i \oplus j = a \oplus b \oplus l \\ &\Leftrightarrow a \oplus i \oplus b \oplus j = a \oplus b \oplus l \\ &\Leftrightarrow (u_{a\oplus i, b\oplus j}, z_{a\oplus b\oplus l}) \in E \\ &\Leftrightarrow (\varphi(u_{i,j}), \varphi(z_l)) \in E.\end{aligned}$$

In any automorphism ϕ with the restrictions $\phi(x_i) = x_{a\oplus i}$ and $\phi(y_i) = y_{b\oplus i}$, the node $\phi(u_{i,j})$ must have edges to $x_{a\oplus i}$ and $y_{b\oplus j}$, but the only node with such connections is $u_{a\oplus i, b\oplus j} = \varphi(u_{i,j})$.

Analogously $\phi(z_i)$ must be connected to $\phi(u_{0,i}) = u_{a,b\oplus i}$, and this implies $\phi(z_i) = z_{a\oplus b\oplus i} = \varphi(z_i)$. This means that φ is the unique automorphism in G^k mapping x_i to $x_{a\oplus i}$ and y_i to $y_{b\oplus i}$. □

We observe that the gadget in Lemma 3.2 for the case $k = 2$ has already been used for a different application in [12]. It is not hard to see that a gadget like the one defined in Lemma 3.2 for $(\mathbb{Z}_k, \oplus)$ can be constructed for any finite Abelian group $G = (A, \circ)$. We mean by this that for any such group a graph whose automorphism group simulates the group operation $\circ$ in the sense of the lemma can be defined.

THEOREM 3.3. *For any $k \geq 2$, GI is hard for $Mod_k L$ under AC^0 many-one reductions.*

Proof. Let $k \geq 2$. We reduce the mod_k circuit value problem to GI. We transform an instance C of the circuit value problem for mod_k circuits into a graph G_C by constructing for every modular gate g_j of C a subgraph like the one described in Lemma 3.2. Moreover, we color the x, y, u, and z nodes of the jth gadget, respectively, with one of the colors $(x, j), (y, j), (u, j)$, and (z, j). Connections between gates are translated in the following way: If the output z of a gate in the circuit is connected to one of the inputs x of another gate, the reduction puts k additional edges connecting (for $i \in \{0, \ldots, k-1\}$) node z_i from the first gate to node x_i from the second gate. For an input variable v^j, k nodes $v_0^j, \ldots, v_{k-1}^j$ are considered in the reduction. The coloring implies that in any automorphism the nodes corresponding to a gate are mapped to nodes from the same gate. Suppose the input variables of the circuit, $v^1, \ldots, v^n$, take values $a_1, \ldots, a_n$. It follows from Lemma 3.2, by induction on the circuit depth, that the output gate z takes value $b \in \{0, \ldots, k-1\}$ if and only if there is an automorphism in G_C mapping v_i^j to $v_{i\oplus a_i}^j$ for all $i = 0, \ldots, k-1$ and $j = 1, \ldots, n$, and mapping z_i to $z_{i\oplus b}$.

All the steps in the reduction can be done locally by an AC^0 circuit. The question of whether the output of the circuit equals $b \in \{0, \ldots, k-1\}$ can be easily reduced to whether two graphs G_b, G_b' are isomorphic, as explained in the preliminaries. In fact this question can be reduced to two graphs pairs $((G, H), (I, J)) \in \mathrm{PGI}$, with G being isomorphic to H if the value of the circuit is b, and I being isomorphic to J otherwise. For this it suffices to define G as G_b, H as G_b', and I and J as the standard OR-function for GI of $\bigcup_{i\neq b}(G_i, G_i')$. □

Observe that the graphs obtained in the reduction have at most k^2 nodes with the same color (the nodes $u_{i,j}$ in any of the gate gadgets). The maximum degree can be reduced to 3. In the above description this does not necessarily hold because of the connection between gates. However, the reduction can be easily modified to achieve degree 3 by adding some extra nodes and arranging the fan-out connections of the gates in a tree-like fashion.

4. Hardness for other complexity classes. In this section we show the hardness of GI for nondeterministic logarithmic space, for $C_=L$, for probabilistic logarithmic space, and for the class DET of problems NC^1 reducible to the determinant. The proofs follow by the modular results, using the Chinese remainder theorem (CRT).

A Chinese remainder representation base is a set $m_1, \ldots, m_n$ of pairwise coprime integers. Let $M = \prod_{i=1}^{n} m_n$. By the CRT, every integer $0 \leq x < M$ is uniquely represented by its Chinese remainder representation $(x_1, \ldots, x_n)$, where $0 \leq x_i < m_i$ and $x_i = x \bmod m_i$. We will consider the base B_n formed by the first n prime numbers.

THEOREM 4.1. *GI is hard for NL under AC^0 many-one reductions.*

Proof. The graph accessibility problem for directed acyclic graphs with fan-in at most 2 is complete for the class NL. We reduce the complement of this set (nonreachability) to GI. The result follows by the closure of NL under complementation [20, 37]. Let $G = (V, E)$ be such a graph, with $|V| = n$ and with two designated nodes s and t. Let P denote the number of paths from s to t in G. Clearly $P \leq 2^n$ and $P = 0$ if and only if for every i between 1 and n it holds that $P \bmod i = 0$.

In the reduction, on input G, an AC^0 circuit, for each i between 1 and n, transforms G into a circuit C_i with addition modulo i gates. The circuits have the property that their outputs coincide with $P \bmod i$ (see the preliminaries). In a second step the reduction transforms the sequence of C_i circuits into a sequence of graphs G_{C_i} (as in the proof of Theorem 3.3) in which there is an automorphism mapping the input nodes according to the inputs of C_i and mapping z_0^i (the node corresponding to the output gate of G_{C_i}) to z_j^i if and only if $P = j \bmod i$. The number of paths from s to t in G is then 0 if and only if for all $i \leq n$ there is an automorphism in G_{C_i} mapping the input nodes G_{C_i} according to the inputs of C_i and mapping z_0^i to itself. This can be easily reduced to GI, as explained in the preliminaries. □

Observe that in the graphs obtained in this reduction, the sizes of the classes of the nodes with the same color are not bounded by a constant, as before, but by n^2.

In fact, we can reduce any logarithmic space counting function to GI. We understand by this that for any function $f \in \#L$ the set

$$A_f = \{\langle x, 0^i \rangle \mid \text{ the } i\text{th bit of } f(x) \text{ is } 1\}$$

is many-one reducible to GI.

For proving this reduction, we need two known results. On the one hand we need the surprising fact that division can be computed by uniform TC^0 circuits[1] [17, 18]. More precisely we need the following part of the mentioned result.

THEOREM 4.2 (see [17, 18]). *There is a DLOGTIME uniform family of TC^0 circuits that, on inputting the Chinese remainder representation $(x_1, \ldots, x_n)$ in base B_n of a number x, outputs the binary representation of x.*

We also need the fact that the result of an NC^1 circuit with fixed values in the input nodes can be encoded as a graph isomorphism question. This follows from an adaptation of the proof of Theorem 3.1 in [22], stating that GI is hard for NC^1 under DLOGTIME uniform AC^0 many-one reductions. For completeness we give a sketch of the proof. The reader is referred to [22] for the details. For technical reasons needed in the proof of Theorem 4.9, we encode the values of the circuit as tuples of graphs $((G, H), (I, J))$ in PGI, with $G \simeq H$ and $I \not\simeq J$ for the encoding of a 1 and with

[1] In fact for our purposes the weaker result—stating that division is in alternating time $O(\log n)$, which was proved in [14]—suffices.

$G \not\simeq H$ and $I \simeq J$ for the encoding of a 0. Recall that PGI was the set of graph tuples $((G,H),(I,J))$ with exactly one of the graphs pairs being isomorphic.

THEOREM 4.3. *Given a uniform family of circuits C_n with logarithmic depth and polynomial size and given n tuples of graphs $((G_i,H_i),(I_i,J_i)) \in PGI$, there is an AC^0 reduction constructing a tuple $((G,H),(I,J)) \in PGI$ with the property that $G \simeq H$ if and only if C_n outputs 1 and $I \simeq J$ if and only if C_n outputs 0, where the ith input to C_n consists of the bit of the Boolean value of the statement $G_i \simeq H_i$.*

Proof. (Sketch.) An NC^1 circuit can be simulated by a balanced DLOGTIME uniform family of circuits with fan-out 1, logarithmic depth, polynomial size, and alternating layers of ANDs and ORs [8]. We show how to transform these expressions to graph tuples. The idea is to construct graph gadgets to simulate the AND and OR connectives in the circuit. Given two tuples $((G_1,H_1),(I_1,J_1))$ and $((G_2,H_2),(I_2,J_2))$ in PGI, consider the graphs $G_\wedge, H_\wedge, I_\wedge$, and $J_\wedge$ in Figure 4.1, where an edge between two graphs represents that each node of the first graph is connected to each node of the second graph. These graphs have the property that $G_\wedge \simeq H_\wedge$ if and only if $G_1 \simeq H_1$ and $G_2 \simeq H_2$. Also $I_\wedge \simeq J_\wedge$ if and only if $G_1 \not\simeq H_1$ or $G_2 \not\simeq H_2$ (in this case $I_1 \simeq J_1$ or $I_2 \simeq J_2$).

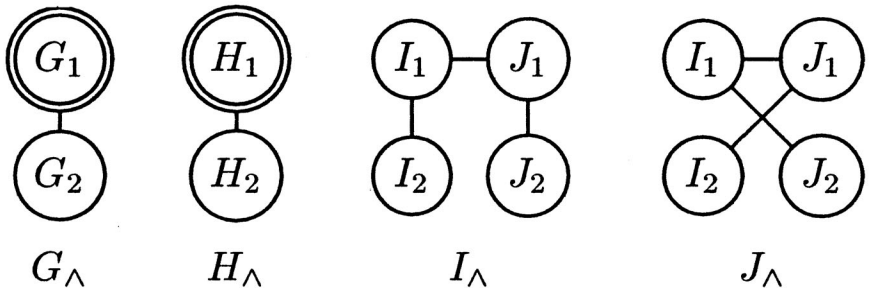


FIG. 4.1. *Tuple $(G_\wedge, H_\wedge, I_\wedge, J_\wedge)$ simulating AND.*

Similarly, the graphs $G_\vee, H_\vee, I_\vee$, and $J_\vee$ from Figure 4.2 have the property that $G_\vee \simeq H_\vee$ if and only if $G_1 \simeq H_1$ or $G_2 \simeq H_2$, and $I_\vee \simeq J_\vee$ if and only if $G_1 \not\simeq H_1$ and $G_2 \not\simeq H_2$. Observe that $((G_\wedge,H_\wedge),(I_\wedge,J_\wedge))$ and $((G_\vee,H_\vee),(I_\vee,J_\vee))$ belong to PGI.

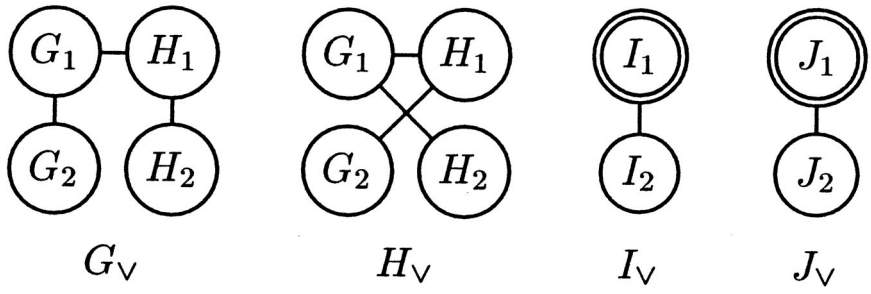


FIG. 4.2. *Tuple $((G_\vee, H_\vee),(I_\vee, J_\vee))$ simulating OR.*

The constructions double the number of nodes of the initial tuples. Notice also that it is easy to simulate a NOT by transforming $((G,H),(I,J))$ to $((I,J),(G,H))$.

A 1 in the circuit is represented by a tuple $((G,H),(I,J))$ with $G \simeq H$, and a 0 by a tuple with $I \simeq J$. Starting from the input nodes the reduction transforms the

nodes of the circuit into graph tuples encoding the values of the circuit gates. Since the circuit has logarithmic depth, the tuples corresponding to the output gate have a polynomial number of nodes. □

We can now show the hardness of GI with respect to #L.

THEOREM 4.4. *Every #L function*[2] *is AC^0 many-one reducible to GI.*

Proof. Let $f \in \#\mathrm{L}$. For some polynomial q, it is possible to construct in AC^0 for $x \in \Sigma^*$ a graph G_x with at most $q(|x|)$ nodes so that $f(x)$ is the number of $s-t$ paths in G_x. Let i be the bit of $f(x)$ we want to reduce to GI, and let $m = q(|x|)$. By Theorem 4.2, in order to compute $f(x)$, it suffices to compute its Chinese remainder representation $(f(x)_1, \ldots, f(x)_m)$ in B_m. Once this is done, $f(x)$ can be computed by an NC^1 circuit.

The Chinese remainder representation can be obtained by computing prime number p_i for every $1 \leq i \leq m$ (this can be done by an NC^1 circuit) and reducing G_x to a circuit with addition gates in $\mathbb{Z}_{p_i}$, as in the proof of Theorem 4.1. The circuits are transformed into p_i graph tuples $((G_j, H_j), (I_j, J_j))$ with the property that in the jth tuple the first two graphs are isomorphic if and only if $f(x) = j - 1 \bmod p_i$. These form a list of $\sum_{i=1}^{m} p_i$ graph tuples and can be considered as an encoding of the Chinese reminder representation of $f(x)$ $(f(x)_1, \ldots, f(x)_m)$ of the form $(w_1, \ldots, w_m)$, where each $w_i \in \{0,1\}^{p_i}$ is formed by 0's with a 1 in position $f(x)_i + 1$. The 0's and 1's in the w_i's are encoded by tuples in PGI.

By Theorem 4.2 it is possible to construct in DLOGTIME a TC^0 (and therefore also an NC^1) circuit that, having as inputs the Chinese reminder representation of $f(x)$, outputs the ith bit of $f(x)$. We can consider the list of graph tuples as the inputs of this circuit.

So far we have shown that there is a uniform AC^0 reduction that on input x computes an NC^1 circuit that outputs the ith bit of $f(x)$ and has its input values encoded as graph tuples in PGI. As done in the proof of Theorem 4.3, an AC^0 reduction can also transform the whole circuit into a single tuple of graphs $((G, H), (I, J))$. G is isomorphic to H or I is isomorphic to J depending on the output of the NC^1 circuit, which coincides with the ith bit of $f(x)$. □

Basically the same proof as the one for the hardness for NL holds for proving hardness for the class $\mathrm{C_{=}L}$. Here instead of checking that the number of paths from s to t is 0, we have to check that this number coincides with some exact threshold $f(G) \leq 2^n$. For this the reduction machine has to compute for each small prime p_i the residue $r_i = f(G) \bmod p_i$ (this can be done in NC^1 [32] and in fact in TC^0 by the mentioned result on division in [18]), and then check whether there is an automorphism that for all i maps z_0^i to $z_{r_i}^i$.

COROLLARY 4.5. *GI is hard for $C_{=}L$ under AC^0 many-one reductions.*

As mentioned in the preliminaries, for a set $L \in \mathrm{PL}$, there is a function $f \in \#\mathrm{L}$ and a polynomial p such that for any input x, $x \in L$ if and only if $f(x) \geq 2^{p(|x|)}$. The next result follows then directly from Theorem 4.4 since an input x belongs to L if and only if at least one of the bits corresponding to positions $\geq p(|x|)$ (starting from the right) in the binary representation of $f(x)$ is a 1.

COROLLARY 4.6. *GI is hard for the class PL under AC^0 many-one reductions.*

The class DET of problems NC^1 Turing reducible to the determinant coincides with $\mathrm{NC}^1(\#\mathrm{L})$ (see, e.g., [2]). Combining Theorems 4.3 and 4.4, we can prove the hardness of GI for DET, which is the strongest known hardness result for GI.

[2] In fact this result also holds for the more powerful class of GapL functions defined in [2].

The proof of this result is based on a simulation of the NC^1 circuit as done in Theorem 4.3, replacing each of the oracle queries to f by a small circuit as in the proof of Theorem 4.4. The main problem here is that while in Theorem 4.4 the input for the #L function to be computed is a binary string x, in the simulation of the NC^1 circuit the input to the oracle calls is not given as a sequence of bits but as a sequence of graph tuples encoding these bits. To deal with this problem we need the following lemma, stating that Theorem 4.4 is also true when the input is encoded as a sequence of tuples.

LEMMA 4.7. *For each function $f \in \#L$ there is a DLOGTIME uniform family $\{C_n\}$ of AC^0 circuits such that, on inputting a sequence of graph tuples $((G_i, H_i), (I_i, J_i))$ in PGI, $1 \leq i \leq n$, of size polynomial in n encoding a binary string $x \in \Sigma^n$, C_n constructs a sequence of tuples $((G'_i, H'_i), (I'_i, J'_i))$ in PGI, $1 \leq i \leq q(n)$, encoding the bits of $f(x)$.*

Proof. Let $f \in \#L$ and $n, k, m \in \mathbb{N}$. From the description of the nondeterministic logarithmic space machine M computing f, the reduction constructs first in AC^0 a graph G_f^n of polynomial size in n related to the configuration graph of M. We can consider that M has a read-only tape for the input and a work tape of logarithmic size. The set of nodes of G_f^n consists of the set of tuples (s, c, p_1, p_2, b), where s is a state of M, c is a possible content of the work tape, p_1 and p_2 are the positions of the tape heads on the input and work tape, respectively, and b is one bit that will be used to encode the content at position p_1 on the input tape. For a concrete input some of these descriptions are not consistent with the input information since b might not be the correct bit at position p_1. Nevertheless we consider the set of all such possible descriptions at this point. This set has polynomial size in n. The set of edges in G_f^n is given by the transition function of M. If the machine can reach from a description $d = (s, c, p_1, p_2, b)$ the configuration (s', c', p'_1, p'_2) in one step, then there is a directed edge in G_f^n from d to $(s', c', p'_1, p'_2, 0)$ and another one from d to $(s', c', p'_1, p'_2, 1)$.

Let x be the input for f encoded by a sequence of graph triplets in PGI. In order to compute whether $f(x) \bmod k$ is congruent with m we will consider that the nodes of G_f^n are addition gates in $\mathbb{Z}_k$ in a polynomial size circuit C. If all the nodes of C would correspond to descriptions consistent with the input, then the output of this circuit would be $f(x) \bmod k$. However, half of the gates in C correspond to inconsistent descriptions and corrupt the final sum. To avoid this problem we use a method that guarantees that the wires coming out of the inconsistent gates always have value 0 and therefore do not contribute to the final sum. This will be done with a new graph gadget. Using first the graph gadgets in section 2.1, circuit C can be transformed into a graph G_C, where each of the mod k gates corresponding to a machine description $d = (s, c, p, p', b)$ is transformed into a subgraph with input nodes $x_0, \ldots, x_{k-1}$ and $y_0, \ldots, y_{k-1}$ and output nodes $z_0, \ldots, z_{k-1}$ in such a way that if there is an automorphism mapping x_l to $x_{l \oplus i}$ and y_l to $y_{l \oplus j}$ in this subgraph, then the automorphism maps z_l to $z_{l \oplus i \oplus j}$ for $i, j, l \in \{0, \ldots, k-1\}$ (Lemma 3.2). The output nodes z_l are then connected with an edge to the input nodes of other gates, nodes $w_0, \ldots, w_{k-1}$ (the nodes of z are connected to as many nodes as the fan-out of the corresponding gate; for simplicity we consider it is just one). Let us suppose that the bit b in description d is 1 (the 0 case is completely analogous) and let $((G_p, H_p), (I_p, J_p))$ be the input tuple in PGI encoding the correct value for the position p in the input x. The gate corresponding to d is a consistent gate if and only if b equals the Boolean value of $G_p \simeq H_p$. To force the inconsistent gates always to propagate a 0 (an automorphism mapping z_0 to itself) the reduction includes between

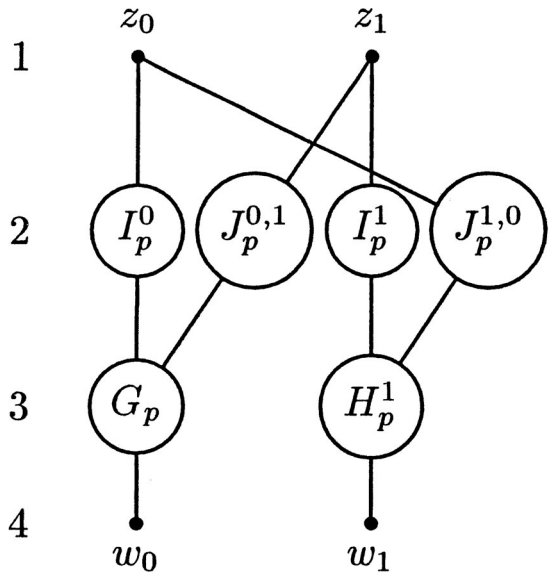


FIG. 4.3. *The graph Gad_2.*

the z and w nodes the following gadget Gad_k, which can be seen in Figure 4.3 for the case $k = 2$. Connections between a node v and a graph in the figure and in the following description of Gad_k mean that there is an edge between v and each of the nodes in the graph.

Subgraph Gad_k can be represented in four levels. Levels 1 and 4 contain the nodes z_i, and w_i, respectively, for $i \in \{0, \ldots, k-1\}$. Level 2 contains for each i a copy I_p^i of I_p and $k-1$ copies of J_p, $J_p^{i,j}$, $j \in \{0, \ldots, k-1\}$, $j \neq i$. Level 3 contains a copy of G_p and for each i in $\{1, \ldots, k-1\}$ a copy H_p^i of H_p. The edges are defined as follows:

- Each node z_i is connected to the graphs I_p^i and to the $k-1$ graphs $J_p^{l,i}$ for $l \neq i$ in the second level.
- Graph G_p in the third level is connected to I_p^0 and to each of the graphs $J_p^{0,j}$ $j \neq 0$, all of them in the second level.
- The graphs H_p^i, $i \neq 0$, in the third level are connected to I_p^i and to $J_p^{i,j}$, $j \neq i$.
- Finally, in the fourth level, node w_0 is connected to G_p and each w_i for $i \neq 0$ is connected to H_p^i.

Gad_k has very nice properties, as can be seen in the next lemma.

LEMMA 4.8. *Subgraph Gad_k has the following properties:*

(1) *If the gate is consistent with the input, that is, if $G_p \simeq H_p$, then for any $c \in \{0, \ldots, k-1\}$ there is an automorphism in Gad_k mapping z_i to $z_{i \oplus c}$ for each i. Such automorphism also maps w_i to $w_{i \oplus c}$.*

(2) *If the gate is inconsistent with the input, that is, if $I_p \simeq J_p$, then for any $c \in \{0, \ldots, k-1\}$ there is an automorphism in Gad_k mapping z_i to $z_{i \oplus c}$ for each i. Such automorphism also maps w_i to w_i.*

Proof. In order to see item 1 of Lemma 4.8, observe that if the automorphism maps z_i to $z_{i \oplus c}$, then the graph I_p^i connected to z_i has to be mapped to one of the graphs connected to $z_{i \oplus c}$, $J_p^{j, i \oplus c}$, or $I_p^{i \oplus c}$. But I_p cannot be mapped to J_p since these graphs are not isomorphic. This implies that in any automorphism all the graphs

I_p in the second level have to be mapped to graphs of type I_p. In particular I_p^i has to be mapped to $I_p^{i\oplus c}$. This means that the graph G_p at the third level has to be mapped to the H_p graph over w_c (this can happen since $G_p \simeq H_p$), and this implies that for all i, w_i has to be mapped to $w_{i\oplus c}$. An automorphism satisfying all these conditions can be defined by mapping for all i, j with $i \neq j$ $J_p^{i,j}$ to $J_p^{i\oplus c, j\oplus c}$ at the second level. Observe that in the case that G_p, H_p, I_p, and J_p are rigid graphs, the described automorphism is the only one mapping z_i to $z_{i\oplus c}$ for each i.

For the proof of item 2 above, observe that in the case that the gate is inconsistent with the input, then the graph G_p at the third level has to be mapped to itself, and therefore the w nodes also have to be mapped to themselves. We have to prove that there is an automorphism with these properties mapping z_i to $z_{i\oplus c}$ for each i. This is clear for $c = 0$. For $c \neq 0$ this automorphism maps in the second level graph I_p^i to $J_p^{i,i\oplus c}$ (this is possible since $I_p \simeq J_p$) and maps $G_p^{i,j}$ to $G_p^{i,j\oplus c}$ if $i \neq j \oplus c$ or to I_p^i in the case that $i = j \oplus c$. The automorphism fixes the third and fourth levels, and again, in the case that G, H, I, and J are rigid graphs, it is the only one mapping z_i to $z_{i\oplus c}$ for each i. □

We continue with the proof of Lemma 4.7. Let G'_C be the graph corresponding to circuit C with the new gadgets on the edges coming out of the gates in C. The above lemma guarantees that inconsistent gates always produce value 0, and therefore the circuit produces the correct value for $f(x)$ mod k. Let $z_0, \ldots, z_{k-1}$ be the output nodes in G'_C corresponding to the output gate of the circuit. By the results in section 2.1, there is an automorphism in G'_C mapping for each i z_i to $z_{i\oplus m}$ if and only if $f(x) \equiv m$ mod k. This property can be encoded by the reduction, using standard methods, into a graph tuple in PGI $((G, H), (I, J))$ satisfying that $G \simeq H$ if $f(x) \equiv m$ mod k, and $G \simeq I$ otherwise. Observe that if the graphs in the tuples have size at most s, then the sizes of the output graphs are at most $p(n)s$ for a polynomial p depending on the machine M. The rest of the proof is exactly as in Theorem 4.4. □

We can now prove the hardness of GI for DET. This result answers positively a question posed by Allender in [1]. Recall that DET can be characterized as $\mathrm{NC}^1(\#\mathrm{L})$, the class of problems computed by an AC^0 uniform family of polynomial size and logarithmic depth circuits with oracle gates to a function f in #L. By convention, an oracle gate querying a string x contributes with $\log(|x| + |f(x)|)$ to the total circuit depth.

THEOREM 4.9. *GI is hard for the class DET under AC^0 many-one reductions.*

Proof. Let L be a set in $\mathrm{NC}^1(\#\mathrm{L})$ and let $\{C_n\}$ be the family of NC^1 circuits computing L with functional oracle queries to a function f in #L.

We want to compute $C_n(x)$ for a string x of length n. The reduction can first transform each oracle gate g into a circuit D_g, as done in Theorem 4.4. Observe that the structure of the circuit computing gate g does not depend on the input bits of g, but just on the number of such bits. D_g computes the query using modular gates as well as AND and OR gates. D_g has polynomial size (in the size of its input) and its depth is not necessarily logarithmic, but the number of levels with AND or OR gates in this circuit is logarithmic in the input size of g. If we count only the depth of the AND and OR gates (the maximum number of such gates in a path from an input to the output gate), C_n with the expanded oracles gates still has logarithmic depth in n since we are dealing with an NC^1 reduction.

Each gate in the circuit C_n with expanded oracle queries can be transformed by the AC^0 reduction into a tuple of four graphs $((G, H), (I, J))$ encoding the value of the

gate as explained before. Using Theorems 4.3 and 4.9, the reduction can construct these tuples for all the levels of the circuit. The graph tuple corresponding to the output gate encodes the result of the circuit computation.

It is only left to show that the size of the graph tuples corresponding to the circuit gates remain of polynomial size in n. The gadgets corresponding to the AND and OR gates increase the size of the graph tuples at most by a factor of 2 in each level, and the number of circuit levels with AND or OR gates is logarithmic in n. The gadgets attached to the modular computations in the query gates increase the sizes of the tuples by a factor of $p(m)$, where m is the size of the query and p is a polynomial. Because C_n computes an NC^1 reduction, in a circuit path with oracle queries with sizes $m_1, \ldots, m_l$, it must hold that the sum of the logarithms of all the query sizes is at most $c \log(n)$ for some constant c. From this it follows that the product of the increasing factors $p(m_i)$ corresponding to all the oracle queries in the path is bounded by a polynomial in n. These facts imply that the sizes of the graph tuples corresponding to every gate in C_n are polynomial in n. □

4.1. Matching is reducible to GI. We mention an interesting connection between the perfect matching problem and GI. The perfect matching problem consists in determining whether a given undirected graph has a perfect matching, that is, a set of edges that contain all the vertices, and such that no two of these edges share a vertex. This problem has been intensively studied, but like GI, it has resisted all classification attempts in terms of completeness in a class. The problem has polynomial time algorithms, and it is known to be in random NC [24, 33]. In [5] it has been proved that for any $k \geq 2$, the perfect matching problem is randomly reducible to a set in $\mathrm{Mod}_k\mathrm{L}$. Together with Theorem 3.3 this implies the following corollary.

COROLLARY 4.10. *The perfect matching problem is reducible to GI under randomized reductions.*

Since the reduction works correctly with probability exponentially close to 1, for each input size n there is a sequence of random choices that can be taken as correct advice in the reduction of all instances of size n. This implies a nonuniform reduction from the perfect matching problem to GI. Moreover, as noted in [3], under a natural hardness hypothesis, the reduction from the perfect matching problem to $\mathrm{Mod}_k\mathrm{L}$ can be derandomized using techniques from [21, 25]. This yields the following corollary.

COROLLARY 4.11. *If there is a set A in $DSPACE(n)$ and $\delta > 0$ with the property that, for all large n, no circuit of size less than $2^{\delta n}$ accepts exactly the strings of length n in A, then perfect matching is included in Mod_kL for any $k \geq 2$, and thus the problem is reducible to GI under AC^0 many-one reductions.*

5. Hardness results for graph automorphism. The graph automorphism problem (GA)—determining whether a given graph has a nontrivial automorphism—is many-one reducible to GI, and it seems to be a slightly easier problem. In this section we show that the proven hardness results for GI hold also for GA. We show first that the hardness for the modular classes can be easily translated to GA.

THEOREM 5.1. *For any $k \geq 2$, GA is hard for Mod_kL under AC^0 many-one reductions.*

Proof. In Theorem 3.3 we transformed a circuit with addition gates in $\mathbb{Z}_k$ and values for the input gates into a graph G having a unique automorphism with certain restrictions (some nodes encoding the input and output values of the circuits had to be mapped in a certain way) if and only if the output value of the circuit is 1. The question of whether G has an automorphism with the desired properties can in turn be transformed into a GI problem by making two copies of G, G_1 and G_2. These

graphs have to include some coloring in the nodes representing the input and output values of the circuit in order to encode the restrictions in the automorphism. Observe that there is at most one isomorphism between G_1 and G_2. From this follows that there is a nontrivial automorphism in $G_1 \cup G_2$ if and only if the output of the original circuit is 1. □

Based on this theorem the proof of Theorem 4.1 can be modified to show hardness of GA for NL.

COROLLARY 5.2. *GA is hard for NL under AC^0 many-one reductions.*

The additional ingredient that is needed to prove the stronger hardness results is the fact that an NC^1 computation can be encoded as a GA question, that is, a version of Theorem 4.3 for GA. A direct translation of this result does not work since GA is not known to have AND-functions. An AND-function for GA is a function that is easy to compute and transforms pairs of graphs into single graphs in such a way that both of the original graphs have nontrivial automorphisms if and only if the final graph has such an automorphism. Dieter van Melkebeek has found a way to avoid this problem.

THEOREM 5.3 (van Melkebeek). *Given a uniform family of circuits C_n with logarithmic depth and polynomial size and given n tuples of rigid graphs $((G,H),(I,J)) \in$ PGI, there is an AC^0 reduction constructing a tuple of rigid graphs $((G,H),(I,J)) \in$ PGI with the property that $G \simeq H$ if and only if C_n outputs 1, and $I \simeq J$ if and only if C_n outputs 0, where the ith input to C_n consists of the bit of the Boolean value of the statement $G_i \simeq H_i$.*

Proof. The proof is like that for Theorem 4.3 simulating the alternating layers of ANDs and ORs of an NC^1 circuit by graph gadgets for the tuples. The main difficulty is preserving the rigidity of the tuple components.

In order to simulate the AND, given two tuples of rigid graphs $((G_1,H_1),(I_1,J_1))$ and $((G_2,H_2),(I_2,J_2))$ in PGI consider the graphs $G_\wedge, H_\wedge, I_\wedge$, and $J_\wedge$ in Figure 5.1. $G_\wedge$ and $H_\wedge$ are defined as the standard AND-function for GI of the G and H graphs, while $I_\wedge$ and $J_\wedge$ are constructed as the OR of (I_1,J_1) and (I_2 AND G_1, J_2 AND H_1).

These graphs have the property that $G_\wedge \simeq H_\wedge$ if and only if $G_1 \simeq H_1$ and $G_2 \simeq H_2$. Also $I_\wedge \simeq J_\wedge$ if and only if $I_1 \simeq J_1$ (and therefore $G_1 \not\simeq H_1$) or $I_2 \simeq J_2$ (in this case $G_2 \not\simeq H_2$ and either $G_1 \simeq H_1$ or $I_1 \simeq J_1$). Observe that if all the graphs in the input tuples are rigid, then $G_\wedge, H_\wedge, I_\wedge$, and $J_\wedge$ are also rigid.

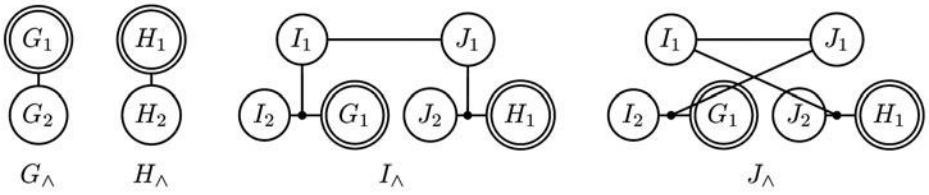


FIG. 5.1. *Tuple $(G_\wedge, H_\wedge, I_\wedge, J_\wedge)$ simulating AND.*

Similarly, the graphs $G_\vee, H_\vee, I_\vee$, and $J_\vee$ from Figure 5.2 have the property that $G_\vee \simeq H_\vee$ if and only if $G_1 \simeq H_1$ or $G_2 \simeq H_2$ and $I_\vee \simeq J_\vee$ if and only if $G_1 \not\simeq H_1$ and $G_2 \not\simeq H_2$. These gadgets simulate, therefore, an OR gate. Moreover, if the all the graphs G_i, H_i, I_i, and J_i are rigid for $i \in \{1,2\}$, then the constructed graphs $G_\vee$, $H_\vee$, $I_\vee$, and $J_\vee$ are also rigid.

Observe that the size of the constructed gadgets is at most $3n$, n being the sum of

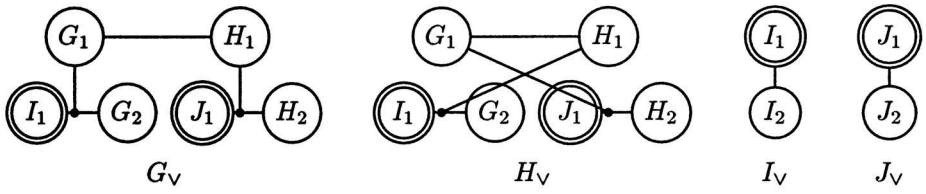


FIG. 5.2. *Tuple* $(G_\vee, H_\vee, I_\vee, J_\vee)$ *simulating OR.*

all the nodes in the input tuples. Because of this fact, for a logarithmic depth circuit C with alternating layers of AND and OR fan-out 1 gates, a tuple of polynomial size rigid graphs $((G,H)(I,J))$ can be constructed such that C has value 1 if and only if $G \simeq H$. Since G and H are rigid, this is equivalent to $G \cup H \in \mathrm{GA}$. □

An immediate consequence of this result is that GA is hard for NC^1. Using this fact and Theorem 5.1, it is now possible to prove the hardness of GA for the class DET. The proof of this result follows exactly the same lines as that for Theorem 4.9, taking into consideration that the graph pairs produced in the reduction from Theorem 3.3 are rigid, and that the gadgets in the proof of Theorem 4.9 also preserve rigidity.

COROLLARY 5.4. *GA is hard for the class DET under* AC^0 *many-one reductions.*

One final observation is that from Theorem 5.1 it follows also that the perfect matching problem is randomly reducible to GA.

Acknowledgments. I would like to thank D. van Melkebeek for providing the proof for Theorem 5.3. He and an anonymous referee suggested also many improvements to the original version of the results. I had several interesting discussions with V. Arvind and J. Köbler that clarified the results of the paper. E. Allender pointed out to me the connection to the determinant. I also would like to thank D. Therien for organizing the McGill Invitational Workshop, where this research started.

REFERENCES

[1] E. ALLENDER, *The division breakthroughs*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 74 (2001), pp. 61–77.

[2] E. ALLENDER AND M. OGIHARA, *Relationships among PL, #L, and the determinant*, RAIRO Inform. Théor. Appl., 30 (1996), pp. 1–21.

[3] E. ALLENDER, K. RHEINHARDT, AND S. ZHOU, *Isolation, matching, and counting: Uniform and nonuniform upper bounds*, J. Comput. System Sci., 59 (1999), pp. 164–181.

[4] C. ÀLVAREZ AND B. JENNER, *A very hard logspace counting class*, Theoret. Comput. Sci., 107 (1993), pp. 3–30.

[5] L. BABAI, A. GÁL, AND A. WIDGERSON, *Superpolynomial lower bounds for monotone span programs*, Combinatorica, 19 (1999), pp. 301–319.

[6] L. BABAI, D. GRIGORYEV, AND D. MOUNT, *Isomorphism of graphs with bounded eigenvalue multiplicity*, in Proceedings of the 14th Annual ACM Symposium on Theory of Computing, 1982, pp. 310–324.

[7] L. BABAI AND E. LUKS, *Canonical labeling of graphs*, in Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 171–183.

[8] D. A. M. BARRINGTON, N. IMMERMAN, AND H. STRAUBING, *On uniformity within* $\mathcal{NC}^1$, J. Comput. System Sci., 41 (1990), pp. 274–306.

[9] R. BOPPANA, J. HASTAD, AND S. ZACHOS, *Does co-NP have short interactive proofs?*, Inform. Process. Lett., 25 (1987), pp. 27–32.

[10] G. BUNTROCK, C. DAMM, U. HERTRAMPF, AND C. MEINEL, *Structure and importance of logspace-MOD-classes*, Math. System Theory, 25 (1992), pp. 223–237.

[11] S. R. Buss, *Alogtime algorithms for tree isomorphism, comparison, and canonization*, in Computational Logic and Proof Theory, Lecture Notes in Comput. Sci. 1289, Springer-Verlag, Berlin, 1997, pp. 18–33.

[12] J. Cai, M. Fürer, and N. Immerman, *An optimal lower bound on the number of variables for graph identifications*, Combinatorica, 12 (1992), pp. 389–410.

[13] S. A. Cook, *A taxonomy of problems with fast parallel algorithms*, Inform. and Control, 64 (1985), pp. 2–22.

[14] A. Chiu, *Complexity of Parallel Arithmetic Using the Chinese Remainder Representation*, Master's thesis, University of Wisconsin, 1995.

[15] A. Chiu, G. Davida, and B. Litow, *Division in logspace uniform* NC^1, Theor. Inform. Appl., 35 (2001), pp. 259–275.

[16] J. Gill, *Computational complexity of probabilistic Turing machines*, SIAM J. Comput., 6 (1977), pp. 675–695.

[17] W. Hesse, *Division is in uniform* TC^0, in Proceedings of the 28th International Colloquium on Automata, Languages and Planning (ICALP), Lecture Notes in Comput. Sci. 2076, Springer-Verlag, Berlin, 2001, pp. 104–114.

[18] W. Hesse, E. Allender, and D. M. Barrington, *Uniform constant-depth threshold circuits for division and iterated multiplication*, J. Comput. System Sci., 65 (2002), pp. 695–716.

[19] J. E. Hopcroft and R. E. Tarjan, *A* V^2 *algorithm for determining isomorphism of planar graphs*, Inform. Process. Lett., 1 (1971), pp. 32–34.

[20] N. Immerman, *Nondeterministic space is closed under complementation*, SIAM J. Comput., 17 (1988), pp. 935–938.

[21] R. Impagliazzo and A. Wigderson, $P = BPP$ *if* E *requires exponential circuits: Derandomizing the XOR lemma*, in Proceedings of the 29th ACM Symposium on Theory of Computing, 1997, pp. 220–229.

[22] B. Jenner, J. Köbler, P. McKenzie, and J. Torán, *Completeness results for graph isomorphism*, J. Comput. System Sci., 66 (2003), pp. 549–566.

[23] B. Jenner, P. McKenzie, and J. Torán, *A note on the hardness of tree isomorphism*, in Proceedings of the 13th IEEE Computational Complexity Conference, 1998, pp. 101–106.

[24] R. Karp, E. Upfal, and A. Wigderson, *Constructing a perfect matching is in random NC*, Combinatorica, 6 (1986), pp. 35–48.

[25] A. R. Klivans and D. van Melkebeek, *Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses*, SIAM J. Comput., 31 (2002), pp. 1501–1526.

[26] J. Köbler, U. Schöning, and J. Torán, *The Graph Isomorphism Problem—Its Structural Complexity*, Prog. Theoret. Comput. Sci., Birkhäuser, Boston, MA, 1993.

[27] J. Köbler and J. Torán, *The complexity of graph isomorphism for colored graphs with color classes of size* 2 *and* 3, in Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2285, Springer-Verlag, Berlin, 2002, pp. 121–132.

[28] S. Lindell, *A logspace algorithm for tree canonization*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 400–404.

[29] E. Luks, *Isomorphism of bounded valence can be tested in polynomial time*, J. Comput. System Sci., 25 (1982), pp. 42–65.

[30] E. Luks, *Parallel algorithms for permutation groups and graph isomorphism*, in Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 292–302.

[31] R. Mathon, *A note on the graph isomorphism counting problem*. Inform. Process. Lett., 8 (1979), pp. 131–132.

[32] P. McKenzie and S. A. Cook, *The parallel complexity of Abelian permutation group problems*, SIAM J. Comput., 16 (1987), pp. 880–909.

[33] K. Mulmuley, U. Vazirani, and V. Vazirani, *Matching is as easy as matrix inversion*, Combinatorica, 7 (1987), pp. 105–113.

[34] W. Ruzzo, *On uniform circuit complexity*, J. Comput. System Sci., 22 (1981), pp. 365–383.

[35] W. Ruzzo, J. Simon, and M. Tompa, *Space bounded hierarchies and probabilistic computations*, J. Comput. System Sci., 28 (1984), pp. 216–230.

[36] U. Schöning, *Graph isomorphism is in the low hierarchy*, J. Comput. System Sci., 37 (1988), pp. 312–323.

[37] R. Szelepcsényi, *The method of forced enumeration for nondeterministic automata*, Acta Inform., 26 (1988), pp. 279–284.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1109–1135

A LAMBDA CALCULUS FOR QUANTUM COMPUTATION*

ANDRÉ VAN TONDER†

Abstract. The classical lambda calculus may be regarded both as a programming language and as a formal algebraic system for reasoning about computation. It provides a computational model equivalent to the Turing machine and continues to be of enormous benefit in the classical theory of computation. We propose that quantum computation, like its classical counterpart, may benefit from a version of the lambda calculus suitable for expressing and reasoning about quantum algorithms. In this paper we develop a quantum lambda calculus as an alternative model of quantum computation, which combines some of the benefits of both the quantum Turing machine and the quantum circuit models. The calculus turns out to be closely related to the linear lambda calculi used in the study of linear logic. We set up a computational model and an equational proof system for this calculus, and we argue that it is equivalent to the quantum Turing machine.

Key words. quantum computation, lambda calculus, linear logic, models of computation

AMS subject classifications. 81P68, 68N18, 68Q10, 03B70

DOI. 10.1137/S0097539703432165

1. Introduction.

There are two main approaches to the theory of quantum computation: the quantum Turing machine, introduced by Benioff [1] and Deutsch [2], and the quantum circuit model, introduced by Deutsch [3]. These two approaches were shown to be essentially equivalent by Yao [4].

The quantum Turing machine provides a fundamental model of quantum computation that may be regarded as a baseline for defining universality. However, reasoning about Turing machines can be a cumbersome process, requiring word-at-a-time thinking while keeping track of complicated machine and tape states. Turing machine programs do not satisfy a simple algebra.

For this reason, the quantum circuit model is more popular in the practical investigation of quantum algorithms. Quantum circuits are visual and compositional and may be manipulated algebraically. However, no single finite quantum circuit is universal. Indeed, Yao's proof of Turing equivalence relies on the concept of uniform circuit families generated by classical computation [4, 5]. To define what we mean by such a circuit family, we need to rely on a separate model of classical computation not described by any finite quantum circuit.

In classical computation, the lambda calculus provides an alternative computational model, equivalent to the Turing machine, which continues to be of enormous utility in the theory of computation, in mathematical logic, and in the study of computer languages and their semantics [6, 7, 8, 9, 10]. Due to its simplicity and expressive power, the lambda calculus has been used as the basis of several powerful computer languages, including Lisp, Scheme, ML, and Haskell [11, 12, 13, 14].

In this article, we propose that quantum computing, like its classical counterpart, may benefit from an alternative computational model based on a version of the lambda calculus suitable for expressing and reasoning about quantum algorithms. We develop such a calculus, which turns out to be closely related to the linear lambda calculi used

*Received by the editors July 24, 2003; accepted for publication (in revised form) March 15, 2004; published electronically July 20, 2004.
http://www.siam.org/journals/sicomp/33-5/43216.html
†Department of Physics, Brown University, Box 1843, Providence, RI 02906 (andre@het.brown.edu).

$t ::=$	*terms:*
x	*variable*
$(\lambda x.\, t)$	*abstraction*
$(t\ t)$	*application*

FIG. 1. *Syntax of the lambda calculus* λ.

in the study of linear logic. We set up its computational model and equational proof system and argue that the computational model is equivalent to the quantum Turing machine.

The quantum lambda calculus combines some of the benefits of both quantum circuits and the quantum Turing machine. The quantum lambda calculus describes functions that may be composed and manipulated algebraically, like quantum circuits. Programs can be algebraically transformed into equivalent programs, and one can solve equations whose unknowns are programs, in much the same way as one solves equations in high school algebra [15]. Unlike quantum circuits, the quantum lambda calculus provides a unified framework that is universal for quantum computation without the need to rely on a separate model of classical computation.

In a practical vein, we show how various known quantum algorithms may be expressed as simple programs in the lambda calculus. Indeed, the calculi described in this paper may be used as a programming language for prototyping quantum algorithms. In fact, the algorithms exhibited in this article were transcribed into Scheme for testing. The simulator, which was also written in Scheme, is available on request from the author.

Since the first version of this paper was written, some progress has been made by the author in devising a typed version, with accompanying denotational semantics, of a fragment of the quantum calculus described here [16].

2. The classical lambda calculus. We begin by providing a reasonably self-contained introduction to concepts and constructions in the classical lambda calculus that will be used in the rest of the paper. The intended audience for this section includes physicists and general computer scientists. The expert may skip this section and refer back as needed.

The classical lambda calculus may be regarded both as a programming language and as a formal algebraic system for reasoning about computation. It was originally introduced by Church in the study of the foundations of mathematics [17, 18]. Church postulated that it provides a universal model of computation, which was later shown by Turing to be equivalent to the Turing machine [19].

As a formal system, the lambda calculus has axioms and rules of inference, and it lends itself to analysis using the language and tools of mathematical logic. Computation may be regarded as guided deduction in this formal system. This provides a directed form of equational reasoning that corresponds to symbolic evaluation of programs via a sequence of algebraic simplifications called reductions [6, 7, 8, 9, 10].

The syntax of the classical untyped lambda calculus λ is as follows. Expressions (also called terms) are constructed recursively from variables x, y, z, ... , parentheses, spaces, the period, and the symbol λ, according to the grammar of Figure 1.

A term of the form $(\lambda x.\, t)$ is called a *functional abstraction*. It represents the

function $x \mapsto t$. For example, the identity function $x \mapsto x$ is written as

$$(\lambda x.\, x)$$

The dummy variable x here is called a bound variable and conforms to the usual rules governing bound variables in mathematical formulas. For example, we identify expressions that differ only in the renaming of bound variables.

A term of the form $(t\ t)$ represents a *function application*. The sole means of computation in the lambda calculus is the operation of applying a function to its argument consistent with the following axiom:

$$((\lambda x.\, t)\ v) = t\,[v/x] \qquad (\beta)$$

Here v denotes a *value*, to be defined shortly. Reading this axiom from left to right defines an algebraic rewrite rule for transforming terms, substituting the argument v in place of the variable x into the function body. This transformation is called beta reduction. We will use the arrow $\longrightarrow$ to indicate one (and sometimes more than one) beta reduction step. A reducible term is called a *redex*.

Unabridged lambda terms can be painful to read. For this reason, we will often introduce abbreviations using the symbol $\equiv$. In addition, we will often omit parentheses according to the convention that nested lambda abstractions associate to the right and applications associate to the left.

Consider the simple program $((\lambda x.\, x)\ \mathsf{apple})$, where apple stands for some term in our language. With the abbreviation $\mathsf{id} \equiv (\lambda x.\, x)$, this may be written more legibly as $(\mathsf{id}\ \mathsf{apple})$, which should evaluate to apple. Indeed, beta reduction gives

$$((\lambda x.\, x)\ \mathsf{apple}) \longrightarrow \mathsf{apple}$$

in a single step.

In general, a computation consists of a sequence of beta reductions executed according to some deterministic strategy until the resulting term cannot be reduced any further, at which point the computation terminates.

A slightly more complicated example, which serves to show how multiple-argument functions can be represented in terms of nested single-argument functions (a technique known as currying), is given by

$$\begin{aligned} \mathsf{apply} &\equiv \lambda f.\, \lambda x.\, (f\ x) \\ &\equiv \lambda f.\, (\lambda x.\, (f\ x)), \end{aligned}$$

which represents a function that applies its first argument f, which should be a function, to its second argument x. Applying the identity function to banana should give banana. To see this, the program $(\mathsf{apply}\ \mathsf{id}\ \mathsf{banana})$, which is shorthand for $((\mathsf{apply}\ \mathsf{id})\ \mathsf{banana})$, is now executed by the following sequence of beta reductions (underlining redexes):

$$\begin{aligned} ((\mathsf{apply}\ \mathsf{id})\ \mathsf{banana}) &\equiv (\underline{((\lambda f.\, (\lambda x.\, (f\ x)))\ \mathsf{id})}\ \mathsf{banana}) \\ &\longrightarrow (\underline{(\lambda x.\, (\mathsf{id}\ x))}\ \mathsf{banana}) \\ &\longrightarrow \underline{(\mathsf{id}\ \mathsf{banana})} \\ &\longrightarrow \mathsf{banana}. \end{aligned}$$

$v ::=$	*values:*
x	*variable*
$(\lambda x.\, t)$	*abstraction value*

FIG. 2. *Values in the call-by-value calculus* λ_v.

$$\frac{t_1 \longrightarrow t_1'}{(t_1\ t_2) \longrightarrow (t_1'\ t_2)} \qquad (app_1)$$

$$\frac{t_2 \longrightarrow t_2'}{(v_1\ t_2) \longrightarrow (v_1\ t_2')} \qquad (app_2)$$

$$(\lambda x.\, t)\ v \longrightarrow t\,[v/x] \qquad (\beta)$$

FIG. 3. *Reduction rules for the call-by-value calculus* λ_v.

Often there is more than one reducible subterm at any given step and a strategy is required to make the process unambiguous. For definiteness, we will use a *call by value* strategy. This works as follows. Abstractions (terms of the form $(\lambda x.\, t)$) are considered *values* and may not be reduced any further. A function application $(t\ t)$ may be reduced only if both the operator and the operand are values. Otherwise the operator and operand must be reduced first. We will call the resulting calculus the call-by-value lambda calculus λ_v. Formally, we state the syntax for values [20, 21] in Figure 2. The reduction rules are listed in Figure 3.

We will denote by $\langle\!\langle t \rangle\!\rangle$ the term, when it exists, obtained by fully reducing t to a value.

We will often use a less cumbersome informal notation when defining functions. For example, the apply function above satisfies the property

$$\mathsf{apply}\ f\ x \longrightarrow (f\ x)$$

under beta reduction. Given this specification, the translation into a lambda term is straightforward.

How do we represent data in the lambda calculus? Since all we have at our disposal are lambda terms, we need a way of encoding data as lambda abstractions with specified properties. There is a technique that can be used for any kind of data structure, which we will illustrate with two examples: natural numbers and lists.

Let us first consider how the natural numbers may be represented. As with any kind of data, we need a way to construct natural numbers and a way to deconstruct them, extracting their constituents.

One possible encoding is as the sequence

$$\underline{0}, \quad \underline{1} \equiv \langle\!\langle \mathsf{succ}\ \underline{0} \rangle\!\rangle, \quad \underline{2} \equiv \langle\!\langle \mathsf{succ}\ \underline{1} \rangle\!\rangle, \quad \ldots,$$

where

$$\underline{0} \equiv \lambda x.\, \lambda y.\, (x\ \mathsf{id}),$$
$$\mathsf{succ} \equiv \lambda n.\, \lambda x.\, \lambda y.\, (y\ n)$$

are the constructors.[1] The above definitions were motivated by the need to be able to define a **case** expression (deconstructor),

$$\textbf{case } t_1 \textbf{ of } (\underline{0} \to t_2,\ \mathsf{succ}\ m \to t_3),$$

which may now be taken as an abbreviation for

$$t_1\ (\lambda z.\, t_2)\ (\lambda m.\, t_3).$$

Here z denotes a variable that does not appear free in t_2. This expression allows us to deconstruct a natural number, extracting the ingredients that went into its construction, i.e., either 0 or its predecessor m. Indeed, it is not difficult to verify the following behavior under beta reduction:

$$\begin{aligned}\textbf{case } \underline{0} \textbf{ of } (\underline{0} \to t_2,\ \mathsf{succ}\ m \to t_3) &\longrightarrow t_2,\\ \textbf{case } \langle\!\langle \mathsf{succ}\ t_0 \rangle\!\rangle \textbf{ of } (\underline{0} \to t_2,\ \mathsf{succ}\ m \to t_3) &\longrightarrow t_3\,[\langle\!\langle t_0 \rangle\!\rangle / m].\end{aligned}$$

As an example, it is now trivial to define the predecessor function (with the convention that $\mathsf{pred}\ \underline{0} = \underline{0}$),

$$\mathsf{pred} \equiv \lambda n.\, \textbf{case } n \textbf{ of } \begin{cases} \underline{0} \to \underline{0} \\ \mathsf{succ}\ m \to m. \end{cases}$$

To program arbitrary computations, we need to verify that the lambda calculus is sufficiently powerful to represent *recursive* functions. Indeed, recursion can be used to represent any kind of iterative or looping computation.

For example, to define addition, we need an expression add which behaves as follows under beta reduction:

$$\mathsf{add}\ m\ n \longrightarrow \textbf{case } m \textbf{ of } \begin{cases} \underline{0} \to n \\ \mathsf{succ}\ k \to \mathsf{add}\ k\ (\mathsf{succ}\ n), \end{cases}$$

where the subterm denoted by add on the left has copied itself into the body of the term on the right-hand side. One of the simplest ways to achieve this is to define [22]

$$\mathsf{add} \equiv (t\ t), \tag{1}$$

where

$$t \equiv \lambda f.\left(\lambda m.\, \lambda n.\, \textbf{case } m \textbf{ of } \begin{cases} \underline{0} \to n \\ \mathsf{succ}\ k \to (f\ f)\ k\ (\mathsf{succ}\ n) \end{cases}\right). \tag{2}$$

In other words, t is an abstraction consisting of the body of the addition function with the combination $(f\ f)$ in the position where add should insert itself after reduction.

[1] Explicitly,

$$\begin{aligned} 0 &\equiv \lambda x.\, \lambda y.\, (x\ (\lambda w.\, w)),\\ 1 &\equiv \lambda x.\, \lambda y.\, (y\ \lambda x.\, \lambda y.\, (x\ (\lambda w.\, w))),\\ 2 &\equiv \lambda x.\, \lambda y.\, (y\ \lambda x.\, \lambda y.\, (y\ \lambda x.\, \lambda y.\, (x\ (\lambda w.\, w))))\\ &\vdots \end{aligned}$$

It is a simple exercise to show that add indeed has the specified behavior under beta reduction. This method can be applied to any recursive function.

The computation of the program ($\mathsf{add}\ \underline{2}\ \underline{2}$) then proceeds via the following sequence of beta reductions:

$$\begin{aligned}
\mathsf{add}\ \underline{2}\ \underline{2} &\equiv \mathsf{add}\ \langle\!\langle \mathsf{succ}\ \underline{1} \rangle\!\rangle\ \underline{2} \\
&\longrightarrow \mathbf{case}\ \langle\!\langle \mathsf{succ}\ \underline{1} \rangle\!\rangle\ \mathbf{of}\ (\underline{0} \to \underline{2},\ \mathsf{succ}\ k \to \mathsf{add}\ k\ (\mathsf{succ}\ \underline{2})) \\
&\longrightarrow \mathsf{add}\ \underline{1}\ \langle\!\langle \mathsf{succ}\ \underline{2} \rangle\!\rangle \equiv \mathsf{add}\ \langle\!\langle \mathsf{succ}\ \underline{0} \rangle\!\rangle\ \underline{3} \\
&\longrightarrow \mathbf{case}\ \langle\!\langle \mathsf{succ}\ \underline{0} \rangle\!\rangle\ \mathbf{of}\ (\underline{0} \to \underline{3},\ \mathsf{succ}\ k \to \mathsf{add}\ k\ (\mathsf{succ}\ \underline{3})) \\
&\longrightarrow \mathsf{add}\ \underline{0}\ \langle\!\langle \mathsf{succ}\ \underline{3} \rangle\!\rangle \equiv \mathsf{add}\ \underline{0}\ \underline{4} \\
&\longrightarrow \mathbf{case}\ \underline{0}\ \mathbf{of}\ (\underline{0} \to \underline{4}, \mathsf{succ}\ k \to \mathsf{add}\ k\ (\mathsf{succ}\ \underline{4})) \\
&\longrightarrow \underline{4}.
\end{aligned}$$

The above technique can be generalized to arbitrary data structures. For example, lists can be represented by the following constructors, which are entirely analogous to those of the natural numbers:

$$\begin{aligned}
() &\equiv \lambda x.\, \lambda y.\, (x\ \mathsf{id}), \\
\mathsf{cons} &\equiv \lambda h.\, \lambda t.\, \lambda x.\, \lambda y.\, ((y\ h)\ t),
\end{aligned}$$

where $()$ denotes the empty list and cons constructs a list consisting of a first (head) element h followed by a list t (the tail) containing the rest of the elements. Again, the above definitions were motivated by the need to be able to define a deconstructor

$$\mathbf{case}\ t_1\ \mathbf{of}\ (() \to t_2,\ \mathsf{cons}\ h\ t \to t_3),$$

which may now be taken as an abbreviation for

$$t_1\ (\lambda z.\, t_2)\ (\lambda h.\, \lambda t.\, t_3).$$

We will often abbreviate $h : t \equiv (\mathsf{cons}\ h\ t)$. We can define tuples in terms of lists as $(x_1, \ldots, x_n) \equiv x_1 : x_2 : \cdots : x_n : ()$. Under beta reduction, we have the behavior

$$\begin{aligned}
\mathbf{case}\ ()\ \mathbf{of}\ (() \to t_2,\ h : t \to t_3) &\longrightarrow t_2, \\
\mathbf{case}\ \langle\!\langle t_0 : t_1 \rangle\!\rangle\ \mathbf{of}\ (() \to t_2,\ h : t \to t_3) &\longrightarrow t_3\,[\langle\!\langle t_0 \rangle\!\rangle / h,\ \langle\!\langle t_1 \rangle\!\rangle / t],
\end{aligned}$$

showing how the case expression may be used to deconstruct the list, extracting its head and tail. To see how these abstractions are used, consider the recursive function

$$\mathsf{map}\ f\ \mathit{list} \longrightarrow \mathbf{case}\ \mathit{list}\ \mathbf{of}\ \begin{cases} () \to () \\ h : t \to (f\ h) : (\mathsf{map}\ f\ t), \end{cases}$$

which takes as input a function and a list and applies the function to each element of the list. The reader may verify that, for example,

$$\mathsf{map}\ \mathsf{double}\ (\underline{4}, \underline{7}, \underline{2}) \longrightarrow (\underline{8}, \underline{14}, \underline{4}), \qquad \mathsf{double} \equiv \lambda x.\, (\mathsf{add}\ x\ x).$$

Finally, we introduce some convenient notation. Since we can represent tuples as lists, we can define functions on tuples using notation such as

$$\lambda(x, y).\, t \equiv \lambda u.\, \mathbf{case}\ u\ \mathbf{of}\ \begin{cases} () \to () \\ \\ x : t' \to \mathbf{case}\ t'\ \mathbf{of}\ \begin{cases} () \to () \\ \\ y : t'' \to t. \end{cases} \end{cases}$$

For example, $(\lambda(x, y).\,(\mathsf{add}\ x\ y))\ (\underline{7}, \underline{7})$ evaluates to $\underline{14}$. It is also useful to have a notation for representing intermediate results. The **let** notation

$$\begin{aligned}\textbf{let}\ (x_1, \ldots, x_n) = (t_1, \ldots, t_n)\ \textbf{in}\ t \\ \equiv (\lambda(x_1, \ldots, x_n).\,t)\ (t_1, \ldots, t_n)\end{aligned}$$

allows us to write terms such as

$$\begin{aligned}&\textbf{let}\ x = \underline{1}\ \textbf{in} \\ &\textbf{let}\ (y, z) = (\underline{2}, \underline{3})\ \textbf{in} \\ &\ \mathsf{add}\ x\ (\mathsf{add}\ y\ z)\end{aligned}$$

which evaluates to $\underline{6}$.

3. A quantum computational model. In this section we will construct a computational model, based on the lambda calculus, suitable for describing quantum computations. The language used will be an adaptation of the classical lambda calculus, extended with a set of quantum primitives. We will denote it by λ_i, where the subscript stands for intermediate. For reasons to be discussed in the next section, this language is not suitable as a formal system. In particular, reduction in λ_i does not correspond to a simple system for equational reasoning. In section 5 we will correct these deficiencies to obtain the full quantum lambda calculus λ_q.

In the classical lambda calculus, beta reduction consumes the program to give the result. At each step, information is discarded, which makes the process irreversible. For quantum computing, we need reduction rules that take computational states to superpositions of states in a way that is unitary and reversible.

Bennett [23] showed that any classical computation can be transformed into a reversible computation. The construction, adapted to our situation, is as follows. Let x denote the term being computed, and let $\beta : x \mapsto \beta(x)$ denote a single beta reduction step. Instead of the noninvertible function β, one considers the function $x \mapsto (x, \beta(x))$, which is invertible on its range. In its simplest version, the computation proceeds as

$$x \mapsto (x, \beta(x)) \mapsto (x, \beta(x), \beta^2(x)) \mapsto (x, \beta(x), \beta^2(x), \beta^3(x)) \mapsto \cdots.$$

More complicated schemes exist that reversibly erase the intermediate steps, saving space at the expense of running time. Although this process does not end by itself, we may observe it and regard the computation as having terminated when $\beta^{n+1}(x) = \beta^n(x)$, at which time we may stop the machine by external intervention.

Although this scheme can be used to reversibly implement computations in the classical lambda calculus, we will soon see that it does not work unmodified in the quantum case.

To represent computations involving qubits, we will add a few constant symbols as additional primitives to our language, as in Figure 4.

The symbols 0 and 1 here are *primitives* and should not be confused with the abbreviations $\underline{0}$ and $\underline{1}$ of the previous section. Additional constants H, S, ..., will denote elementary gate operations on qubits. These should include symbols for a universal set of elementary quantum gates [3, 24, 25]. For example, the set consisting of the Hadamard gate H, the phase gate S, the $\pi/8$ gate R_3, and the controlled-not gate *cnot* is universal [25, 5]. Additional primitives, such as the Pauli gates X, Y, and Z, may be added for convenience.

$t ::=$	*terms:*
x	*variable*
$(\lambda x.\, t)$	*abstraction*
$(t\ t)$	*application*
c	*constant*
$c ::=$	*constants:*
$0 \mid 1 \mid H \mid S \mid R_3 \mid \mathit{cnot} \mid X \mid Y \mid Z \mid \ldots$	

FIG. 4. *Syntax of the intermediate language* λ_i.

We now allow the state of a computation to be a quantum superposition of terms in this language. As a model, one may imagine lambda terms encoded as strings of symbols on the tape of a quantum Turing machine.

As a first example, consider an initial state written in ket notation as

$$|(H\ 0)\rangle\,.$$

We would like to choose the transition rules of the quantum computer in such a way that this string will evaluate to the Hadamard operator applied to $|0\rangle$, which should give the superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ of the states $|0\rangle$ and $|1\rangle$ containing unit-length strings. The candidate reduction rule

$$\begin{aligned} |(H\ 0)\rangle &\longrightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,, \\ |(H\ 1)\rangle &\longrightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned}$$

is not reversible. To make it reversible, we first try the same trick as in the classical case,

$$\begin{aligned} |(H\ 0)\rangle &\longrightarrow \frac{1}{\sqrt{2}}(|(H\ 0);\, 0\rangle + |(H\ 0);\, 1\rangle) \\ &= |(H\ 0)\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,, \end{aligned}$$

where we have factored out the common substring. The semicolon denotes string concatenation. In this simple example, the answer indeed factors out on the right. However, notice what happens if we apply this method to the term

$$\begin{aligned} |(H\ (H\ 0))\rangle &\longrightarrow \frac{1}{\sqrt{2}}(|(H\ (H\ 0));\, (H\ 0)\rangle + |(H\ (H\ 0));\, (H\ 1)\rangle) \\ &\longrightarrow \frac{1}{2}\,|(H\ (H\ 0))\rangle \otimes \\ &\qquad \otimes \Big(|(H\ 0);\, 0\rangle + |(H\ 0);\, 1\rangle + |(H\ 1);\, 0\rangle - |(H\ 1);\, 1\rangle\Big). \end{aligned}$$

Here the answer does not factor out. The fully reduced rightmost term is entangled with the intermediate term in the history.

$v ::=$	*values:*
x	*variable*
c	*constant*
$(\lambda x.\, t)$	*abstraction value*

FIG. 5. *Values in the intermediate language* λ_i.

Note, however, that this scheme keeps more information than necessary. For reversibility, it is sufficient to record at each step only which subterm has been reduced and the operation that has been applied to it. We may encode this in our example as follows (to be formalized below):

$$\begin{aligned}|(H\ (H\ 0))\rangle &\longrightarrow \frac{1}{\sqrt{2}}\left(|(\_\ (H\ \_));\ (H\ 0)\rangle + |(\_\ (H\ \_));\ (H\ 1)\rangle\right)\\ &\longrightarrow \frac{1}{2}\,|(\_\ (H\ \_))\rangle \otimes\\ &\qquad \otimes \Big(|(H\ \_);\ 0\rangle + |(H\ \_);\ 1\rangle + |(H\ \_);\ 0\rangle - |(H\ \_);\ 1\rangle\Big)\\ &= |(\_\ (H\ \_));\ (H\ \_)\rangle \otimes |0\rangle\,.\end{aligned}$$

Here, at each step we have replaced subterms that do not need to be recorded by the constant placeholder symbol _. Now the answer does indeed factor out on the right as required, consistent with $H^2\,|0\rangle = |0\rangle$. It is also clear that at each step we have kept enough information to reconstruct the previous step, thus ensuring reversibility.

The computational model may now be formalized with the following rules: First, we extend our definition of values to include constants as in Figure 5. The computational state is taken to be a quantum superposition of sequences of the form

$$h_1;\ \ldots\ ;\ h_n;\ t,$$

where $h_1;\ \ldots\ ;\ h_n$ will be called the history track and t will be called the computational register. The classical subset of the transition rules is shown in Figure 6.

In these rules $\mathcal{H}$ denotes the (possibly empty) history track, and $\overline{t}_x$ is obtained from t by recursively replacing all subterms that do not contain x with the placeholder symbol and keeping x. More formally,

$$\begin{aligned}\overline{t}_x &\equiv \_ \qquad \text{if } x \text{ not free in } t,\\ \overline{(\lambda y.\, t)}_x &\equiv (\_.\,\overline{t}_x),\\ \overline{(t\ t')}_x &\equiv (\overline{t}_x\ \overline{t'}_x),\\ \overline{x}_x &\equiv x.\end{aligned} \tag{3}$$

These rules are sufficient to make classical computations reversible, provided that lambda terms that differ only by renaming of bound variables have been identified. In this regard, we note here that in a quantum Turing machine model, it is possible to represent terms on the tape of the quantum Turing machine in an unambiguous way (e.g., using De Bruijn indices instead of bound variables) [7, 8].

$$\frac{t_1 \longrightarrow h_1;\, t_1'}{\mathcal{H};\, (t_1\ t_2) \longrightarrow \mathcal{H};\, (h_1\ \_);\, (t_1'\ t_2)} \qquad (app_1)$$

$$\frac{t_2 \longrightarrow h_2;\, t_2'}{\mathcal{H};\, (v_1\ t_2) \longrightarrow \mathcal{H};\, (\_\ h_2);\, (v_1\ t_2')} \qquad (app_2)$$

$$\mathcal{H};\, ((\lambda x.\, t)\ v) \longrightarrow \mathcal{H};\, ((\lambda x.\, \overline{t}_x)\ \_);\, t\,[v/x] \qquad (\beta_1) \text{ if } x \text{ appears free in } t$$

$$\mathcal{H};\, ((\lambda x.\, t)\ v) \longrightarrow \mathcal{H};\, ((\lambda x.\, \_)\ v);\, t \qquad (\beta_2) \text{ if } x \text{ not free in } t$$

$$\mathcal{H};\, t \longrightarrow \mathcal{H};\, \_;\, t \qquad (Id) \text{ otherwise}$$

FIG. 6. *Operational model for the classical subset of* λ_i.

$$|\mathcal{H};\, (H\ 0)\rangle \longrightarrow |\mathcal{H};\, (H\ \_)\rangle \otimes \frac{1}{\sqrt{2}}\,(|0\rangle + |1\rangle)$$

$$|\mathcal{H};\, (H\ 1)\rangle \longrightarrow |\mathcal{H};\, (H\ \_)\rangle \otimes \frac{1}{\sqrt{2}}\,(|0\rangle - |1\rangle)$$

FIG. 7. *Operational model for* H.

Here is an example computation:

$$\begin{aligned}
|((\mathsf{apply}\ \mathsf{id})\ \mathsf{banana})\rangle &\equiv (((\lambda f.\,(\lambda x.\,(f\ x)))\ (\lambda z.\, z))\ \mathsf{banana}) \\
&\longrightarrow |(((\lambda f.\,(_.\,(f\ _)))\ _)\ _);\, ((\lambda x.\,((\lambda z.\, z)\ x))\ \mathsf{banana})\rangle \\
&\longrightarrow |(((\lambda f.\,(_.\,(f\ _)))\ _)\ _);\, ((\lambda x.\,(_\ x))\ _);\, ((\lambda z.\, z)\ \mathsf{banana})\rangle \\
&\longrightarrow |(((\lambda f.\,(_.\,(f\ _)))\ _)\ _);\, ((\lambda x.\,(_\ x))\ _);\, ((\lambda z.\, z)\ _);\, \mathsf{banana}\rangle \\
&\longrightarrow |(((\lambda f.\,(_.\,(f\ _)))\ _)\ _);\, ((\lambda x.\,(_\ x))\ _);\, ((\lambda z.\, z)\ _);\, _;\, \mathsf{banana}\rangle \\
&\longrightarrow |(((\lambda f.\,(_.\,(f\ _)))\ _)\ _);\, ((\lambda x.\,(_\ x))\ _);\, ((\lambda z.\, z)\ _);\, _;\, _;\, \mathsf{banana}\rangle \\
&\longrightarrow \cdots .
\end{aligned}$$

At each step, just enough information is kept to reconstruct the previous step. Although in this particular example termination can be tested by observing and comparing the last expression in the history with _, in general we do not have a well-defined criterion for termination in the calculus λ_i, because the state may involve a superposition of several computational histories, some of which have terminated and others not. Thus, to observe termination would potentially disturb the state. This problem will be solved in the quantum calculus λ_q of section 5.

In addition, we have some extra reduction rules involving the quantum gate symbols such as those provided in Figure 7.

The rules for quantum primitives are summarized in Figure 8.

Here c_U denotes any one of the quantum primitive symbols and U the corresponding unitary transformation, while ϕ stands for 0 or 1 in the case of single-bit operators, or one of $(0,0)$, $(0,1)$, $(1,0)$ or $(1,1)$ in the case of two-bit operators. For example,

$$|(cnot\ (1,0))\rangle \longrightarrow |(cnot\ \_);\, (1,1)\rangle\,.$$

$$|\mathcal{H};\ (c_U\ \phi)\rangle \longrightarrow |\mathcal{H};\ (c_U\ \_)\rangle \otimes U\,|\phi\rangle \qquad (U)$$

FIG. 8. *Operational model for the quantum primitives of* λ_i.

4. Toward an equational theory. While the language λ_i constructed in the previous section can be used to describe quantum computations, reduction in λ_i does not correspond to a simple system for equational reasoning. This makes λ_i unsuitable as a formal proof system for quantum computation. We will discuss the problem in this section and resolve it in the next with the introduction of the quantum lambda calculus λ_q.

In the classical lambda calculus, program evaluation through beta reduction can be regarded as a directed form of equational reasoning consistent with the axiom

$$(\lambda x.\,t)\ v = t\,[v/x] \qquad (\beta).$$

Indeed, the classical lambda calculus provides both a model of computation and a formal system for reasoning about functions, a property we would like to keep in the quantum case.

To understand the difficulty, notice what happens when a function application discards its argument (in other words, the argument does not appear in the function body). For example,

$$|((\lambda x.\,\mathsf{apple})\ \mathsf{banana})\rangle \longrightarrow |((\lambda x.\,\_)\ \mathsf{banana});\ \mathsf{apple}\rangle\,.$$

We see that to maintain reversibility, a record of the argument banana is kept in the history. Restricting our attention to the computational register, we see that its evolution is consistent with replacing the original expression with an equal expression according to the axiom (β). In other words, in this example reduction is consistent with equational reasoning.

However, we run into problems when the discarded subterm is in a quantum superposition with respect to the computational basis. For example, consider the reduction of

$$\begin{aligned}|(\lambda x.\,0)\ (H\ 0)\rangle &\longrightarrow |(\_\ (H\ \_)\rangle \otimes \frac{1}{\sqrt{2}}\Big(|(\lambda x.\,0)\ 0\rangle + |(\lambda x.\,0)\ 1\rangle\Big)\\ &\longrightarrow |(\_\ (H\ \_)\rangle \otimes \frac{1}{\sqrt{2}}\Big(|(\lambda x.\,\_)\ 0\rangle + |(\lambda x.\,\_)\ 1\rangle\Big) \otimes |0\rangle\,.\end{aligned}$$

In the second step, a discarded subterm in a superposition is saved in the history and the computational register becomes $|0\rangle$. However, if we were to apply the axiom (β) to the contents of the computational register, we would get the equation

$$\frac{1}{\sqrt{2}}\Big(|(\lambda x.\,0)\ 0\rangle + |(\lambda x.\,0)\ 1\rangle\Big) = \sqrt{2}\ |0\rangle\,,$$

which is invalid since the right-hand side is not a legal normalized state.

As a second example, consider the following computation, where the inner func-

tion discards its argument x:

$$
\begin{aligned}
&|((\lambda y.\,((\lambda x.\,y)\ y))\ (H\ 0))\rangle \\
&\quad\longrightarrow \frac{1}{\sqrt{2}}\,|(\_\ (H\ \_))\rangle \otimes \Big(|((\lambda y.\,((\lambda x.\,y)\ y))\ 0)\rangle + |((\lambda y.\,((\lambda x.\,y)\ y))\ 1)\rangle\Big) \\
&\quad\longrightarrow \frac{1}{\sqrt{2}}\,|(\_\ (H\ \_));\ ((\lambda y.\,((\_.\,y)\ y))\ \_)\rangle \otimes \\
&\qquad\qquad \otimes \Big(|((\lambda x.\,0)\ 0)\rangle + |((\lambda x.\,1)\ 1)\rangle\Big) \\
&\quad\longrightarrow \frac{1}{\sqrt{2}}\,|(\_\ (H\ \_));\ ((\lambda y.\,((\_.\,y)\ y))\ \_)\rangle \otimes \\
&\qquad\qquad \otimes \Big(|((\lambda x.\,\_)\ 0);\ 0\rangle + |((\lambda x.\,\_)\ 1);\ 1\rangle\Big).
\end{aligned}
$$

Now the computational register is entangled with the last expression in the history. Ignoring the history, the computational register would be in a mixed state with density matrix

$$\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}.$$

However, an attempt to apply the equational axiom (β) to the contents of the computational register would give

$$\frac{1}{\sqrt{2}}\Big(|((\lambda x.\,0)\ 0)\rangle + |((\lambda x.\,1)\ 1)\rangle\Big) = \frac{1}{\sqrt{2}}\,(|0\rangle + |1\rangle)\,,$$

which is clearly inconsistent.

5. A quantum lambda calculus. We will resolve the shortcomings of the language λ_i by developing a quantum lambda calculus λ_q which has a consistent equational theory. This section will be somewhat heavier on the formalities, and the reader who wishes to see some concrete examples may skip ahead to section 7 after reading the introductory paragraphs.

The previous discussion suggests that the problems with equational reasoning in the presence of quantum operations can be avoided by preventing functions from discarding arguments that may be in a superposition with respect to the computational basis.

Let us call a subexpression definite with respect to the computational basis if it is textually the same in all branches of the superposition. For example, in the state

$$\frac{1}{\sqrt{2}}\Big(|(\lambda x.\,0)\ 0\rangle + |(\lambda x.\,0)\ 1\rangle\Big),$$

the subexpression $(\lambda x.\,0)$ is definite, whereas the argument $\frac{1}{2}\,(|0\rangle + |1\rangle)$ is nondefinite. Definite subexpressions may be thought of as a classical resource. They can be observed without affecting the state of the computation. On the other hand, nondefinite subexpressions represent purely quantum resources.

To avoid the problems pointed out in the previous section, we seek a calculus that will keep track of whether an argument is definite or nondefinite and that will make it impossible to write a function that discards a nondefinite resource. Calculi that are

$t ::=$	*terms:*
x	*variable*
$(\lambda x.\, t)$	*abstraction*
$(t\ t)$	*application*
c	*constant*
$!t$	*nonlinear term*
$(\lambda !x.\, t)$	*nonlinear abstraction*
$c ::=$	*constants:*
$0 \mid 1 \mid H \mid S \mid R_3 \mid cnot \mid X \mid Y \mid Z \mid \ldots$	

FIG. 9. *Syntax of the quantum calculus* λ_q.

resource sensitive, known as linear lambda calculi, have been studied intensively in recent years [26, 27, 28, 29]. So-called typed linear lambda calculi are very closely related to the field of linear logic [30, 31]. Linear logic is a resource-sensitive logic where, for example, certain assumptions may be used only once in the course of a derivation.

For our purposes it will be sufficient to study a simple untyped linear calculus. The syntax is a fragment of the one introduced in [27], extended with quantum operations as in Figure 9.

Here terms of the form $!t$ are called nonlinear. Nonlinear terms will be guaranteed to be definite with respect to the computational basis and may be thought of physically as classical strings of symbols that may be discarded and duplicated at will. On the other hand, linear terms may be nondefinite, possibly containing embedded qubits in superpositions with respect to the computational basis. Abstractions of the form $(\lambda !x.\, t)$ denote functions of nonlinear arguments. In an abstraction of the form $(\lambda x.\, t)$, the argument is called linear.

A functional abstraction may use a nonlinear argument any number of times in its body, or not at all. On the other hand, a linear argument must appear exactly once in the function body (hence the name linear).

To enforce these rules, we require terms to be well formed. This corresponds to the constraint that linear arguments appear linearly in a function body and that all free variables appearing in a term $!t$ refer to nonlinear variables [28]. In the following examples, the terms in the left column are well formed while those in the right column are ill formed:[2]

$(\lambda !x.\, 0)$	$(\lambda x.\, 0)$
$(\lambda x.\, x)$	$(\lambda x.\, !x)$
$(\lambda !x.\, (x\ x))$	$(\lambda x.\, (x\ x))$
$(\lambda y.\, (\lambda !x.\, y))$	$(\lambda y.\, (\lambda x.\, y))$
$(\lambda !y.\, !(\lambda !x.\, y))(\lambda y.\, !(\lambda !x.\, y))$.	

[2]Notice that while well-formedness guarantees that linear resources will be used appropriately, it does not guarantee that terms are meaningful. For example, the term $(\lambda y.\, (\lambda\, !z.0)\ y)$ is well formed, but it may or may not get stuck at run time, according to the operational model of Figure 12, when applied to a linear or nonlinear argument, respectively. A typed calculus would be needed to specify which terms can be legally substituted for y. For recent progress in this direction, see [16].

$$\frac{}{\vdash c} \quad \textit{Const}$$

$$\frac{}{x \vdash x} \quad \textit{Id}$$

$$\frac{!x_1, \ldots, !x_n \vdash t}{!x_1, \ldots, !x_n \vdash !t} \quad \textit{Pomotion}$$

$$\frac{\Gamma, x \vdash t}{\Gamma, !x \vdash t} \quad \textit{Dereliction}$$

$$\frac{\Gamma, !x, !y \vdash t}{\Gamma, !z \vdash t\,[z/x, z/y]} \quad \textit{Contraction}$$

$$\frac{\Gamma \vdash t}{\Gamma, !x \vdash t} \quad \textit{Weakening}$$

$$\frac{\Gamma, x \vdash t}{\Gamma \vdash (\lambda x.\, t)} \quad \multimap\textit{-I}$$

$$\frac{\Gamma, !x \vdash t}{\Gamma \vdash (\lambda !x.\, t)} \quad \to\textit{-I}$$

$$\frac{\Gamma \vdash t_1 \quad \Delta \vdash t_2}{\Gamma, \Delta \vdash (t_1\ t_2)} \quad \multimap\textit{-E}$$

FIG. 10. *Rules for well-formed terms in the quantum calculus* λ_q.

Well-formedness is a property that can be checked syntactically. For completeness, we formally state the rules for well-formedness [28], which the reader satisfied with the above informal characterization may skip, in Figure 10.

These rules may be related to the typed linear calculi described in [28, 27] by erasing the type annotations from the typing rules of the latter. Here Γ and Δ denote *contexts*, which are sets containing linearity assumptions of the form x and $!x$, where each variable x is distinct. If Γ and Δ are contexts with no variables in common, then Γ, Δ denotes their union. For example, the rule $\multimap$-E implicitly assumes that $\Gamma \cap \Delta = \emptyset$. Rules may be read as follows. For example, the promotion rule says that if t is a well-formed term under the assumption that x_1 to x_n are nonlinear, then $!t$ is a well-formed term under the same assumption. The condition $\Gamma \cap \Delta = \emptyset$ in ($\multimap$-E) ensures that a linear variable can appear only once in the body of a formula. The weakening and ($\to$-I) rules allow a function to discard a nonlinear argument, whereas the contraction and ($\multimap$-E) rules allow us to duplicate a nonlinear argument any number of times in the body of a function.

The well-formedness constraint prevents us from writing a function that discards a linear argument. However, this is not sufficient to prevent unsafe computations without further specification of the substitution order. To see this, consider the expression $((\lambda !x.\, 0)\ !(H\ 0))$, which is well formed. The problem is that we are allowed to use ! to promote the expression $(H\ 0)$ to a nonlinear value, which can then be discarded. If we were allowed to reduce the subterm $(H\ 0)$ first, equational reasoning

$v ::=$	*values:*
x	*variable*
c	*constant*
$(\lambda x.\, t)$	*linear abstraction*
$(\lambda !x.\, t)$	*nonlinear abstraction*
$!t$	*!-suspension*

FIG. 11. *Values in the quantum calculus* λ_q.

$$\frac{t_1 \longrightarrow h_1;\, t_1'}{\mathcal{H};\, (t_1\ t_2) \longrightarrow \mathcal{H};\, (h_1\ \_);\, (t_1'\ t_2)} \qquad (app_1)$$

$$\frac{t_2 \longrightarrow h_2;\, t_2'}{\mathcal{H};\, (v_1\ t_2) \longrightarrow \mathcal{H};\, (\_\ h_2);\, (v_1\ t_2')} \qquad (app_2)$$

$$\mathcal{H};\, ((\lambda x.\, t)\ v) \longrightarrow \mathcal{H};\, ((\lambda x.\, \overline{t}_x)\ \_);\, t\,[v/x] \qquad (\beta)$$

$$\mathcal{H};\, ((\lambda !x.\, t)\ !t') \longrightarrow \mathcal{H};\, ((\lambda !x.\, \overline{t}_x)\ \_);\, t\,[t'/x] \qquad (!\beta_1) \text{ if } x \text{ appears free in } t$$

$$\mathcal{H};\, ((\lambda !x.\, t)\ !t') \longrightarrow \mathcal{H};\, ((\lambda !x.\, \_)\ !t');\, t \qquad (!\beta_2) \text{ if } x \text{ not free in } t$$

$$|\mathcal{H};\, (c_U\ \phi)\rangle \longrightarrow |\mathcal{H};\, (c_U\ \_)\rangle \otimes U\,|\phi\rangle \qquad (U)$$

$$\mathcal{H};\, t \longrightarrow \mathcal{H};\, \_;\, t \qquad (Id) \text{ otherwise}$$

FIG. 12. *Operational model for the quantum lambda calculus* λ_q.

would give

$$\begin{aligned} |((\lambda !x.\, 0)\ !(H\ 0))\rangle &= \frac{1}{\sqrt{2}}\Big(|((\lambda !x.\, 0)\ !0)\rangle + |((\lambda !x.\, 0)\ !1)\rangle\Big) \\ &= \sqrt{2}\,|0\rangle\,, \end{aligned}$$

which is an invalid equation since the last line is not a valid normalized state. On the other hand, if we consider $!(H\ 0)$ as an irreducible value, we may use beta reduction immediately to obtain

$$|((\lambda !x.\, 0)\ !(H\ 0))\rangle = |0\rangle\,,$$

which is a valid result, since we are discarding the unevaluated expression $!(H\ 0)$, which is definite.

To prevent terms of the form $!t$ from being evaluated, we follow Abramsky [26] and extend our definition of values as in Figure 11.

The computational model is described in Figure 12,[3] where $\overline{t}$ is defined as in (3).

According to these rules, quantum superpositions can be created only by evaluating terms containing quantum primitives. The result of applying a quantum gate is a

[3]See [26, 32, 33] for related operational interpretations of linear lambda calculi. Our evaluation model recomputes !-closures (see [32]).

linear value, not preceded by a !. As we prove below, there is no way to include such a linear value in a nonlinear subterm. It follows that subterms that may be quantum nondefinite will never be discarded since, by $(!\beta_1)$ and $(!\beta_2)$, nonlinear functions can be applied only to nonlinear terms.

Note that when a nonlinear function encounters a linear argument, it simply gets stuck. More precisely, the rule (Id) applies.

The above reduction rules may create superpositions. However, such superpositions are not arbitrary. Indeed, terms in a superposition will differ only in positions containing the constants 0 and 1. Otherwise they will have the same shape. We may formalize this by defining two terms to be *congruent* if they coincide symbol by symbol except possibly in positions containing 0 or 1. Lemma 5.1 then follows.

LEMMA 5.1. *All terms in a superposition obtained via a reduction sequence from a definite initial term are congruent.*

Proof. The proof is by a simple induction on the length of the reduction sequence, analyzing the reduction rules case by case. □

Another case-by-case induction argument may be used to prove that reduction preserves well-formedness.

LEMMA 5.2. *If t is well formed and $|\mathcal{H}; t\rangle \longrightarrow \sum_i c_i\, |\mathcal{H}'_i; t'_i\rangle$, then all terms t'_i appearing in the resulting superposition are well formed.*[4]

Because terms appearing in a superposition have the same shape, it makes sense to talk about specific subterms of the expression in the computational register. We can therefore formulate the following lemma.

LEMMA 5.3. *Starting from a definite initial term, any* !*-suspension subterm occurring during reduction is definite with respect to the computational basis.*

Proof. This follows by induction on the length of the reduction sequence. The initial term is definite by assumption. Assume that the lemma holds after n steps. We have argued that all terms in a superposition obtained from a definite initial term are congruent. They therefore have the same structure of subterms, and the same reduction rule applies to them all. Since we have argued that these terms are well formed, there are then three ways in which we may obtain a !-suspension subterm after $n+1$ steps. First, the suspension may not be part of the redex, in which case it is included unmodified in the resulting expression. Second, it may be the result of beta reduction of an application of the form

$$(\lambda x.\,(\cdots x\cdots))\ (\cdots !t\cdots),$$

where $!t$ is definite by the induction assumption. The result is $(\cdots(\cdots !t\cdots)\cdots)$, where $!t$ has been copied without modification. Third, it may be the result of beta reduction of an application of the form

$$(\lambda !x.\ \cdots !(\cdots x\cdots)\cdots)\ !t,$$

where $!t$ and $!(\cdots x\cdots)$ are definite by the induction assumption. This creates a suspension $!(\cdots t\cdots)$, which is definite because all its subterms are definite. This completes the proof. □

It is worth pointing out that we cannot create possibly nondefinite suspensions by reducing terms like

$$(\lambda x.\ \cdots !(\cdots x\cdots)\cdots)\ (H\ 0)$$

because x is linear, which implies that $!(\cdots x\cdots)$ is not a well-formed subterm.

[4]Thanks to one of the referees for suggesting improvements in the exposition of this section.

$$\frac{t_1 \longrightarrow t_1'}{(t_1\ t_2) \longrightarrow (t_1'\ t_2)} \qquad (app_1)$$

$$\frac{t_2 \longrightarrow t_2'}{(v_1\ t_2) \longrightarrow (v_1\ t_2')} \qquad (app_2)$$

$$(\lambda x.\, t)\ v \longrightarrow t\,[v/x] \qquad (\beta)$$

$$(\lambda !x.\, t)\ !t' \longrightarrow t\,[t'/x] \qquad (!\beta)$$

$$|c_U\ \phi\rangle \longrightarrow U\,|\phi\rangle \qquad (U)$$

FIG. 13. *Reduction rules for the quantum calculus* λ_q.

LEMMA 5.4. *Given a definite initial term, the contents of the history track remains definite throughout reduction.*

Proof. We have argued that all terms in a superposition obtained from a definite initial term are congruent. They therefore have the same structure of subterms, and the same reduction rule applies to them all. Since our reduction rules allow only !-suspensions to be discarded (which saves a copy in the history), and since !-suspensions are always definite by the previous lemma, the result follows by induction on the length of the reduction sequence. □

Since both the history and the shape of the term in the computational register remain definite throughout reduction, we can state the following conclusion.

COROLLARY 5.5. *Termination can be tested without disturbing the computation by observing the last term in the history. When this term becomes equal to the placeholder _, the result can be read off from the computational register.*

Not only is termination a "classical" property, but so is the entire shape of the term itself, i.e., term shapes can be implemented in classical memory, and only their data slots need to point to qubits on a quantum device.[5]

The fact that the history remains definite in λ_q eliminates the specific impediments to setting up an equational theory that were pointed out in the previous section. Indeed, since the state of the computation is now always guaranteed to be a direct product $|\mathcal{H}\rangle \otimes |c\rangle$ of the history $|\mathcal{H}\rangle$ and the computational register $|c\rangle$, reduction can never lead to a computational register $|c\rangle$ that is in a mixed state. In addition, since $|\mathcal{H}\rangle$ remains definite, the restriction of the reduction rules to the computational register will preserve the normalization. We are therefore led to the following theorem.

THEOREM 5.6. *In the quantum calculus* λ_q*, the evolution of the computational register is governed by the reduction rules of Figure* 13.

Proof. This easily follows from a case-by-case analysis of the computational rules of Figure 12.

For example, consider the rule $(!\beta_2)$ applied to a state of the form $|\mathcal{H}\rangle \otimes |c\rangle$, where $|c\rangle$ is a normalized superposition of the form

$$\sum_i c_i\ |(\cdots_i\ ((\lambda !x.\, t_i)\ !t')\ \cdots_i)\rangle\,,$$

in which, by Lemma 5.1, all terms have the same structure and, by Lemma 5.3, the

[5] I would like to thank one of the referees for suggesting this improved formulation.

subterm $!t'$ does not depend on i. This then reduces to

$$|\mathcal{H}\rangle \otimes \sum_i c_i \, |(\_\_\_((\lambda!x.\,\_)\;!t')\_\_\_);\, (\cdots_i\; t_i\,[t'/x]\;\cdots_i)\rangle$$
$$= |\mathcal{H};\, (\_\_\_((\lambda!x.\,\_)\;!t')\_\_\_)\rangle \otimes \sum_i c_i \, |(\cdots_i\; t_i\,[t'/x]\;\cdots_i)\rangle\,,$$

where the computational register is in the normalized state $\sum_i c_i \, |(\cdots_i\; t_i\,[t'/x]\;\cdots_i)\rangle$, consistent with applying the reduction rule $(\lambda!x.\,t_i)\;!t' \longrightarrow t_i\,[t'/x]$ to its contents. □

The big win is that we now have a simple set of reduction rules that can be used to reason about the computation without having to keep track of the history.

To define an equational theory for this calculus, we will simply define a notion of equality that is compatible with the reduction rules of Figure 13. Intuitively, reduction should be understood as a simple algebraic operation of replacing subterms with equal subterms. However, we need to take into account that reduction may not take place inside !-suspensions.

We therefore need to introduce algebraic rules governing just when we can replace subterms in an expression with equal subterms [20, 21]. One way to do that is to introduce the notion of a *term context*, which are expressions with a hole $[\,]$ in place of a subexpression,

$$C,\, C_i ::= [\,] \mid (t\;C) \mid (C\;t) \mid (\lambda x.\,C) \mid (\lambda!x.\,C) \mid \sum_i c_i\, C_i,$$

where the last term denotes a superposition of shape-congruent contexts. It is important to note that there are no contexts of the form $!(\cdots[\,]\cdots)$. As a result, subterms preceded by ! will be opaque in the sense that we will not be able to perform substitutions under the ! sign.

DEFINITION 5.7. *The equational theory of λ_q is the least equivalence relation $=$ containing the reduction relation ($\longrightarrow$) of Figure* 13 *and which is closed under substitution in term contexts* [20, 21]. *In other words,*

$$\frac{t_1 = t_2}{C[t_1] = C[t_2]} \qquad (\textit{subst}),$$

where C is an arbitrary term context and $C[t]$ denotes the textual replacement of the hole in C by the term t, extended by linearity to superpositions of congruent terms and congruent contexts, i.e., for $t = \sum_i c_i\, t_i$, we define $C[t] = \sum_i c_i\, C[t_i]$ and for $C = \sum_i c_i\, C_i$, we define $C[t] = \sum_i c_i\, C_i[t]$.

An alternative way to present the equational theory is by listing a set of axioms and rules of inference as in Figure 14.

These rules should again be understood as extending via linearity to congruent superpositions of terms. In this formulation, the rules (*app*), (λ_1), and (λ_2) are together equivalent to the term context substitution rule (*subst*) above. Again, there is no rule that permits substitutions inside !-suspensions.

THEOREM 5.8. *In the quantum lambda calculus, the evolution of the computational register proceeds by replacing terms by equal terms according to the equational theory of Definition* 5.7.

Proof. This is true by construction. □

$$t = t \qquad (\mathit{refl})$$

$$\frac{t_1 = t_2}{t_2 = t_1} \qquad (\mathit{sym})$$

$$\frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3} \qquad (\mathit{trans})$$

$$\frac{t_1 = t_2 \quad t_3 = t_4}{(t_1\ t_3) = (t_2\ t_4)} \qquad (\mathit{app})$$

$$\frac{t_1 = t_2}{\lambda x.\, t_1 = \lambda x.\, t_2} \qquad (\lambda_1)$$

$$\frac{t_1 = t_2}{\lambda !x.\, t_1 = \lambda !x.\, t_2} \qquad (\lambda_2)$$

$$(\lambda x.\, t)\ v = t\,[v/x] \qquad (\beta)$$

$$(\lambda !x.\, t)\ !t' = t\,[t'/x] \qquad (!\beta)$$

$$|(c_U\ \phi)\rangle = U\,|\phi\rangle \qquad (U)$$

FIG. 14. *Equational proof system for the quantum calculus* λ_q.

6. Recursion and a fixed-point operator. Recursive functions may be defined in the calculus λ_q in a way analogous to that described in section 2. We simply replace $(t\ t)$ in (1) with $(t\ !t)$, where now $t \equiv \lambda !f.\,(\cdots\,(f\ !f)\cdots)$.

Here we describe a related approach based on so-called fixed-point combinators. A fixed-point operator suitable for the linear lambda calculus is given by the following adaptation of the classical Turing combinator:

$$\begin{aligned}\mathsf{fix} \equiv (\ &(\lambda !u.\, \lambda !f.\, (f\ !((u\ !u)\ !f))) \\ &!(\lambda !u.\, \lambda !f.\, (f\ !((u\ !u)\ !f)))).\end{aligned}$$

It is easy to check that under reduction

$$\mathsf{fix}\ !t \longrightarrow t\ !(\mathsf{fix}\ !t),$$

where the !-suspension prevents further reduction of the term in brackets. Recursive functions can be defined as follows. If

$$t \equiv \lambda !f.\, u,$$

then it easily follows that

$$\mathsf{fix}\ !t \longrightarrow u\,[(\mathsf{fix}\ !t)/f].$$

In other words, $\mathsf{fix}\ !t$ copies itself into the body u of t under reduction, as required for recursion.

7. Examples of algorithms. We are now ready to formulate some algorithms in the quantum lambda calculus. First, we reproduce some classical constructions, now decorated with the proper nonlinearity annotations.

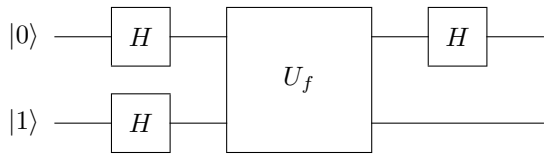


FIG. 15. *Deutsch's algorithm.*

First, we introduce list constructors that will enable us to build lists of linear values (qubits or structures containing qubits),

$$\begin{aligned} () &\equiv \lambda!x.\, \lambda!y.\, (x\ \mathsf{id}),\\ \mathsf{cons} &\equiv \lambda h.\, \lambda t.\, \lambda!x.\, \lambda!y.\, ((y\ h)\ t), \end{aligned}$$

with abbreviations $h : t \equiv (\mathsf{cons}\ h\ t)$ and $(x_1, \ldots, x_n) \equiv x_1 : x_2 : \cdots : x_n : ()$ as before. Since the arguments $!x$ and $!y$ above are nonlinear, we need to redefine our **case** abbreviation as follows:

$$\mathbf{case}\ t_1\ \mathbf{of}\ (() \to t_2,\ h : t \to t_3)$$

now stands for

$$t_1\ !(\lambda!z.\, t_2)\ !(\lambda h.\, \lambda t.\, t_3).$$

Deutsch's algorithm [1, 5] can be very simply expressed as follows:

$$\begin{aligned} \mathsf{deutsch}\ U_f \longrightarrow\ & \mathbf{let}\ (x, y) = U_f\ ((H\ 0), (H\ 1))\ \mathbf{in}\\ & ((H\ x), y) \end{aligned}$$

(see Figure 15). Here the argument U_f is assumed to be a function that takes (x, y) to $(x, y \oplus f(x))$, where f is some (unknown) function of one bit. For example, if f is the identity function, then we should take U_f to be *cnot*. Indeed, the reader may check that

$$|\mathsf{deutsch}\ \mathit{cnot}\rangle \longrightarrow |1\rangle \otimes \frac{1}{2}\Big(|0\rangle - |1\rangle\Big),$$

where the first bit $1 = f(0) \oplus f(1)$ indicates that the function is balanced, as required.

Let us write a simple expression that creates an EPR pair:

$$\mathsf{epr} \equiv \mathit{cnot}\ ((H\ 0), 0).$$

The quantum teleportation gate array with deferred measurement (see Figure 16) [34, 5] can easily be translated into the following code. We create an EPR pair and pass the first EPR qubit, along with the unknown qubit x to be teleported, to Alice. The outcome (x', y') of Alice's computation then gets sent to Bob, who has access to the second EPR qubit e_2,

$$\begin{aligned} \mathsf{teleport}\ x \longrightarrow\ & \mathbf{let}\ (e_1, e_2) = \mathsf{epr}\ \mathbf{in}\\ & \mathbf{let}\ (x', y') = \mathsf{alice}\ (x, e_1)\ \mathbf{in}\\ & \quad \mathsf{bob}\ (x', y', e_2). \end{aligned}$$

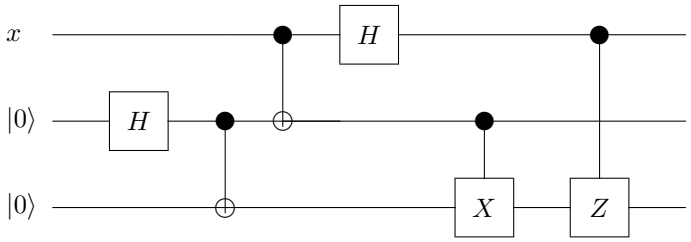


FIG. 16. *Quantum teleportation.*

Here

$$\mathsf{alice}\ (x, e_1) \longrightarrow\ \mathbf{let}\ (x', y') = \mathit{cnot}\ (x, e_1)\ \mathbf{in}\ ((H\ x'), y')$$

and

$$\begin{aligned}\mathsf{bob}\ (x', y', e_2) \longrightarrow\ & \mathbf{let}\ (y'', e_2') = cX\ (y', e_2)\ \mathbf{in} \\ & \mathbf{let}\ (x'', e_2'') = cZ\ (x', e_2')\ \mathbf{in} \\ & (x'', y'', e_2'').\end{aligned}$$

The outcome of the computation consists of the list of three qubits (x'', y'', e_2''). The teleported qubit is e_2'', but notice how linearity requires us to keep the other two qubits in the answer. The reader may check that throughout the computation, linear arguments are used exactly once. Implementing the conditional operations cX and cZ in terms of the primitive constants is left as an easy exercise.

Given recursion and lists, the map function, which applies a given function f to each element of a list, may be defined as

$$\mathsf{map}\ !f\ \mathit{list} \longrightarrow \mathbf{case}\ \mathit{list}\ \mathbf{of}\ \begin{cases} () \to () \\ h : t \to (f\ h) : (\mathsf{map}\ !f\ t) \end{cases}.$$

The arguments *list*, h, and t may refer to qubits or data structures containing qubits and are therefore chosen linear. The expression is well formed because *list*, h, and t are each used exactly once.

It is now trivial to define a program that computes a uniform superposition of a list of qubits by applying the Hadamard gate to each qubit in the list:

$$H^{\otimes n}\ \mathit{list} \longrightarrow \mathsf{map}\ !H\ \mathit{list}.$$

For example, we may evaluate

$$\big|H^{\otimes n}\ (0,0)\big\rangle \longrightarrow \frac{1}{2}\big(\,|(0,0)\rangle + |(0,1)\rangle + |(1,0)\rangle + |(1,1)\rangle\,\big).$$

Note that the well-formedness conditions may be somewhat subtle, as the following example illustrates. A naive attempt at defining an append function that concatenates two linear lists,

$$\mathsf{append}\ x\ y \longrightarrow \mathbf{case}\ x\ \mathbf{of}\ \begin{cases} () \to y \\ h : t \to h : (\mathsf{append}\ t\ y), \end{cases}$$

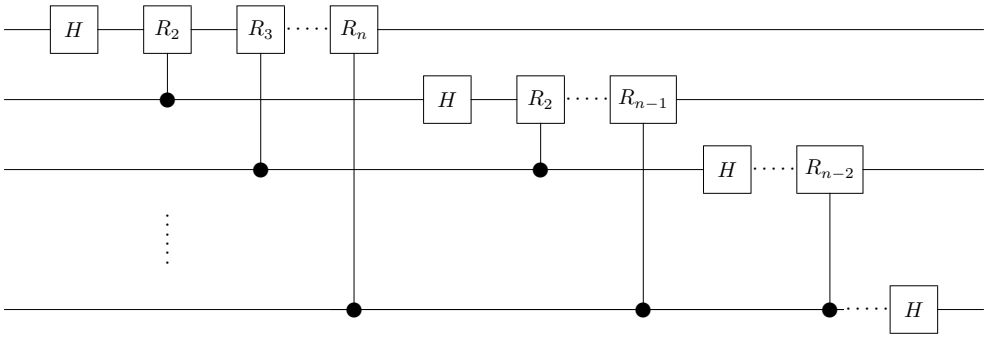


FIG. 17. *The quantum Fourier transform (without reversal).*

fails to be well formed. The problem can be seen by expanding the **case** abbreviation

$$x\ !(\lambda z.\, y)\ !(\lambda h.\, \lambda t.\, (h : (\mathsf{append}\ t\ y))).$$

Since y is a linear variable, we may not promote the λ subterms to nonlinear values with the prepended !. An alternative definition that does work is

$$\mathsf{append}\ x\ y \longrightarrow \left(\mathbf{case}\ x\ \mathbf{of}\ \begin{cases} () \to (\lambda u.\, u) \\ h : t \to \lambda u.\, (h : (\mathsf{append}\ t\ u)) \end{cases}\right)\ y.$$

Next we define a reverse function

$$\mathsf{reverse}\ \mathit{list} \longrightarrow \mathbf{case}\ \mathit{list}\ \mathbf{of}\ \begin{cases} () \to () \\ h : t \to \mathsf{append}\ (\mathsf{reverse}\ t)\ (h). \end{cases}$$

The quantum Fourier transform [35, 36, 37, 38] can now be defined as a direct translation of the corresponding quantum circuit [5] as follows:

$$\mathsf{fourier}\ \mathit{list} \longrightarrow \mathsf{reverse}\ \mathsf{fourier}'\ \mathit{list},$$

where

$$\mathsf{fourier}'\ \mathit{list} \longrightarrow \mathbf{case}\ \mathit{list}\ \mathbf{of}\ \begin{cases} () \to () \\ h : t \to\ \ \mathbf{let}\ h' : t' = \mathsf{phases}\ (H\ h)\ t\ !\underline{2}\ \mathbf{in} \\ \qquad\qquad\qquad h' : (\mathsf{fourier}'\ t') \end{cases}$$

recursively applies the appropriate conditional phase operations to the first qubit in the list, using the helper function

$$\begin{aligned} &\mathsf{phases}\ \mathit{target}\ \mathit{controls}\ !n \\ &\longrightarrow \mathbf{case}\ \mathit{controls}\ \mathbf{of}\ \begin{cases} () \to (\mathit{target}) \\ \mathit{control} : t \to\ \ \mathbf{let}\ (\mathit{control}', \mathit{target}') \\ \qquad\qquad\qquad\quad = (cR\ !n)\ (\mathit{control}, \mathit{target})\ \mathbf{in} \\ \qquad\qquad\qquad \mathbf{let}\ \mathit{target}'' : t' \\ \qquad\qquad\qquad\quad = \mathsf{phases}\ \mathit{target}'\ t\ !(\mathsf{succ}\ n)\ \mathbf{in} \\ \qquad\qquad\qquad\ \ \mathit{target}'' : \mathit{control}' : t'. \end{cases} \end{aligned}$$

Here $(cR\ !n)$ composes an appropriate combination of elementary gates to implement a conditional phase operation with phase $2\pi i/2^n$. Since this is essentially a classical computation and depends on the particular set of primitive constants chosen, we will not write it out here.

Note that we have assumed that the classical construction of the natural numbers may be adapted to the quantum lambda calculus. That this is possible for all classical constructions follows from the fact that there is an embedding of the classical lambda calculus into the linear lambda calculus, as shown in formula (4).

8. Relating λ_q to quantum Turing machines. In this section we will sketch a proof of the following theorem, leaving a more rigorous analysis to future work.

THEOREM 8.1. *The computational model provided by the quantum lambda calculus λ_q is equivalent to the quantum Turing machine.*

Proof. First, we argue that the quantum lambda calculus λ_q may be efficiently simulated on a quantum Turing machine.

In λ_q the current state of the computation consists of a superposition of term sequences of the form $\mathcal{H}; t$, which may be encoded as strings of symbols on the tape of the quantum Turing machine. By Lemma 5.1, term sequences in different branches of the superposition are congruent, and the same reduction rule will apply for all branches at each time step. The subset of λ_q not involving quantum operations consists of a set of reversible classical rewritings, which can be unitarily and efficiently implemented on a quantum Turing machine by [2, 39, 40]. The fragment involving quantum operators again involves simple classical rewritings followed by a unitary transformation involving one or two symbols on the tape. Once again, the methods of [2, 39, 40] may be used to construct a quantum Turing machine that can execute these transformations. This completes the proof of the first half of the equivalence.

Next we argue that a quantum Turing machine can be efficiently simulated by the calculus λ_q.

Yao shows in [4] that for any quantum Turing machine T, there is quantum circuit $C_{n,t}$ that efficiently simulates T on inputs of size n after t steps. The circuit family $C_{n,t}$ may be efficiently constructed via a classical computation. But λ_q is universal for classical computation. This follows from the fact that the classical call-by-value lambda calculus may be embedded in λ_q via the following translation, adapted from [28]:

$$\begin{aligned}(t_1\ t_2)^* &= ((\lambda!z.\, z)\ t_1^*)\ t_2^*,\\ x^* &= !x,\\ (\lambda x.\, t)^* &= !(\lambda!x.\, t^*).\end{aligned} \tag{4}$$

So, given the specification of a quantum Turing machine and an input of length n, a classical computation in λ_q first constructs a representation of the appropriate quantum circuit family $C_{n,t}$. It then follows the circuit diagram and applies the appropriate quantum operations one by one to the input. Since λ_q has primitive quantum operations available corresponding to a universal set of quantum gates, this proves the second half of the equivalence. □

9. Related work. In a series of papers, H. Baker [41, 42, 43] develops an untyped linear language based on Lisp. His language is similar to the classical fragment of the lambda calculus developed in the current article. It served as the initial inspiration for the linear approach followed here.

Ideas stemming from linear logic have been used previously by Abramsky in the study of classical reversible computation [44].

One of the earlier attempts at formulating a language for quantum computation was G. Baker's Qgol [45]. Its implementation (which remained incomplete) used so-called uniqueness types (similar but not identical to our linear variables) for quantum objects [46]. The language is not universal for quantum computation.

The language QCL, developed by Ömer, is described in [47, 48]. QCL is an imperative language with classical control structures combined with special operations on quantum registers. It provides facilities for inverting quantum functions and for scratch space management. No formal program calculus is provided. A simulator is publicly available.

Another imperative language, based on C++, is the Q language developed by Bettelli, Calarco, and Serafini [49]. As in the case of QCL, no formal calculus is provided. A simulator is also available.

A more theoretical approach is taken by Selinger in his description of the functional language QPL [50]. This language has both a graphical and a textual representation. A formal semantics is provided.

The imperative language qGCL, developed by Sanders and Zuliani [51], is based on Dijkstra's guarded command language. It has a formal semantics and proof system.

A previous attempt to construct a lambda calculus for quantum computation is described by Maymin in [52]. However, his calculus appears to be strictly stronger than the quantum Turing machine [53]. It seems to go beyond quantum mechanics in that it does not appear to have a unitary and reversible operational model, instead relying on a more general class of transformations. It is an open question whether the calculus is physically realizable.

A seminar by Wehr [54] suggests that linear logic may be useful in constructing a calculus for quantum computation within the mathematical framework of Chu spaces. However, the author stops short of developing such a calculus.

Abramsky and Coecke describe a realization of a model of multiplicative linear logic via the quantum processes of entangling and deentangling by means of typed projectors. They briefly discuss how these processes can be represented as terms of an affine lambda calculus [55].

10. Conclusion. In this article we developed a lambda calculus λ_q suitable for expressing and reasoning about quantum algorithms. We discussed both its computational model and its equational proof system. We argued that the resulting calculus provides a computational model equivalent to the quantum Turing machine and is therefore universal for quantum computation.

There are many possible directions for future work. The proof of Turing equivalence should be fleshed out. Formal issues relating to consistency and semantics need to be addressed further. While our computational model provides an operational semantics, the problem of providing a denotational semantics is open. The formalism of [56] may be useful in this regard.

In this article, the introduction of a linear calculus was motivated by requiring consistency of its operational model with equational reasoning. The fact that linear arguments, denoting quantum resources, may not be duplicated suggests a separate motivation for linearity, not addressed here, based on the no-cloning theorem [57, 58].

While our calculus is untyped, it would be interesting to investigate typed linear calculi with quantum primitives and, via the Curry–Howard correspondence, the corresponding generalizations of linear logic [59, 60]. We might mention that there have been prior attempts to relate linear logic to quantum mechanics, starting with a suggestion by Girard [30, 61, 62].

On the practical side, the calculi described in this paper may be used as a programming language for prototyping quantum algorithms. Indeed, the algorithms exhibited in this article were transcribed into Scheme for testing. The simulator, which was also written in Scheme, is available on request from the author.

It is our hope that the field of quantum computation, like its classical counterpart, may benefit from the insights provided by the alternative computational model provided by the quantum lambda calculus.

Note. Since the first version of this paper was written, some progress has been made by the author in devising a typed version, with accompanying denotational semantics, of a fragment of the quantum calculus described here [16].

Acknowledgments. I would like to thank Prof. Antal Jevicki and the Brown University physics department for their support. I would also like to thank the two anonymous referees for their brilliant, detailed, and thoughtful comments leading to various corrections and improvements in the exposition.

REFERENCES

[1] P. Benioff, *The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines*, J. Stat. Phys., 22 (1980), pp. 563–591.

[2] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proc. Roy. Soc. London A, 400 (1985), pp. 97–117.

[3] D. Deutsch, *Quantum computational networks*, Proc. Roy. Soc. London A, 439 (1989), pp. 553–558.

[4] A. Yao, *Quantum circuit complexity*, in Proceedings of the 34th Annual Symposium on the Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 352–361.

[5] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.

[6] H. P. Barendregt, *The Lambda Calculus*, North–Holland, Dordrecht, Netherlands, 1984.

[7] J. C. Mitchell, *Foundations of Programming Languages*, MIT Press, Cambridge, MA, 1996.

[8] B. C. Pierce, *Types and Programming Languages*, MIT Press, Cambridge, MA, 2002.

[9] R. E. Davis, *Truth, Deduction, and Computation: Logic and Semantics for Computer Science*, Computer Science Press, New York, 1989.

[10] C. A . Gunter, *Semantics of Programming Languages: Structures and Techniques*, MIT Press, Cambridge, MA, 1992.

[11] J. McCarthy, *Recursive functions of symbolic expressions and their computation by machine, Part* I, Commun. ACM, 3 (1960), pp. 184–195.

[12] D. P. Friedman, M. Wand, and C. T. Haynes, *Essentials of Programming Languages*, MIT Press, Cambridge, MA, 1992.

[13] L. C. Paulson, *ML for the Working Programmer*, Cambridge University Press, Cambridge, UK, 1996.

[14] P. Hudak, *The Haskell School of Expression*, Cambridge University Press, Cambridge, UK, 2000.

[15] J. Backus, *Can programming be liberated from the Von Neumann style? A functional style and its algebra of programs*, Commun. ACM, 21, 1978.

[16] A. van Tonder, *Quantum computation, categorical semantics, and linear logic*, ArXiv.org e-print archive: arXiv:quant-ph/0312174 (2003).

[17] A. Church, *An unsolvable problem in elementary number theory*, Amer. J. Math., 58 (1936), pp. 354–363.

[18] A. Church, *The Calculi of Lambda Vonversion*, Princeton University Press, Princeton, NJ, 1941.

[19] A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc., 42 (1936), pp. 230–265; corrections in Proc. London Math. Soc., 43 (1937), pp. 544–546.

[20] G. Plotkin, *Call-by-name, call-by-value and the λ-calculus*, Theoret. Comput. Sci., 1 (1976), pp. 125–159.

[21] M. Felleisen and R. Hieb, *A revised report on the syntactic theories of sequential control and state*, Theoret. Comput. Sci., 103 (1992), pp. 235–271.
[22] O. Kiselyov, *Many Faces of the Fixed-Point Combinator*, http://okmij.org/ftp/Computation/fixed-point-combinators.html (Oct. 1999).
[23] C. H. Bennett, *Logical reversibility of computation*, IBM J. Res. Develop., 17 (1973), pp. 525–532.
[24] A. Barenco, C. H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Schor, T. Sleator, J. Smolin, and H. Weinfurter, *Elementary gates for quantum computation*, Phys. Rev. A, 52 (1995), pp. 3457–3467.
[25] P. O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan, *On Universal and Fault-Tolerant Quantum Computing*, arXiv:quant-ph/9906054 (1999), http://www.arXiv.org.
[26] S. Abramsky, *Computational interpretations of linear logic*, Theoret. Comput. Sci., 111 (1993), pp. 3–57.
[27] P. Wadler, *A syntax for linear logic*, in Mathematical Foundations of Programming Semantics: 9th International Conference, New Orleans, Proceedings 802, Springer-Verlag, New York, 1993, pp. 513–529.
[28] J. Maraist, M. Odersky, D. Turner, and P. Wadler, *Call-by-name, call-by-value, call-by-need, and the linear lambda calculus*, in 11th International Conference on the Mathematical Foundations of Programming Semantics, New Orleans, 1995.
[29] R. A. G. Seely, *Linear logic, *-autonomous categories, and cofree coalgebras*, in Categories in Computer Science and Logic, AMS Contemporary Mathematics 92, AMS, Providence, RI, 1989.
[30] J.-Y. Girard, *Linear logic*, Theoret. Comput. Sci., 50 (1987), pp. 1–102.
[31] P. Wadler, *A taste of linear logic*, in Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science, Gdánsk, Springer-Verlag, New York, 1993.
[32] D. N. Turner and P. Wadler, *Operational interpretations of linear logic*, Theoret. Comput. Sci., 227 (1999), pp. 231–248.
[33] J. Chirimar, C. A. Gunter, and J. G. Riecke, *Reference counting as a computational interpretation of linear logic*, J. Funct. Programming, 6 (1996), pp. 195–244.
[34] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. Wootters, *Teleporting an unknown quantum state via dual classical and EPR channels*, Phys. Rev. Lett., 70 (1993), pp. 1895–1899.
[35] D. Coppersmith, *An Approximate Fourier Transform Useful in Quantum Factoring*, IBM Research Report RC 19642, IBM, White Plains, NY, 1994.
[36] A. Ekert and R. Jozsa, *Shor's quantum algorithm for factorizing numbers*, Rev. Mod. Phys., 68 (1996), pp. 733–753.
[37] P. W. Shor, *Algorithms for quantum computation: Discrete log and factoring*, in Proceedings of the 35th IEEE FOCS, 1994, pp. 124–134.
[38] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.
[39] E. Bernstein and U. Vazirani, *Quantum complexity theory*, SIAM J. Comput., 26 (1997), pp. 1411–1473.
[40] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and weaknesses of quantum computing*, SIAM J. Comput., 26 (1997), pp. 1510–1523.
[41] H. G. Baker, *Lively linear Lisp—Look Ma, no garbage!*, ACM Sigplan Notices, 27 (1992), pp. 89–98.
[42] H. G. Baker, *A Linear Logic quicksort*, ACM Sigplan Notices, 29 (1994), pp. 13–18.
[43] H. G. Baker, *Use-once variables and linear objects—storage management, reflection and multi-threading*, ACM Sigplan Notices, 30 (1995), pp. 45–52.
[44] S. Abramsky, *A Structural Approach to Reversible Computation*, Research Report RR-01-09, Programming Research Group, Oxford University, Oxford, UK, 2001.
[45] G. D. Baker, *Qgol: A System for Simulating Quantum Computations: Theory, Implementation, and Insights*, Honors thesis, Macquarie University, New South Wales, Australia, 1996.
[46] E. Barendsen and S. Smetsers, *Conventional and Uniqueness Typing in Graph Rewrite Systems*, Technical report CSI-R9328, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, 1993.
[47] B. Ömer, *A Procedural Formalism for Quantum Computing*, M.S. thesis, Technical University, Vienna, 1998; also available online from http://tph.tuwien.ac.at/∼oemer/qcl.html.
[48] B. Ömer, *Classical Concepts in Quantum Programming*, arXiv:quant-ph/0211100 (2002),

http://www.arXiv.org.
[49] S. Bettelli, T. Calarco, and L. Serafini, *Towards an architecture for quantum programming*, Eur. Phys. J. D, 25 (2003), pp. 181–200.
[50] P. Selinger, *Towards a quantum programming language*, Math. Structures Comput. Sci., to appear.
[51] J. W. Sanders and P. Zuliani, *Quantum programming*, in Mathematics of Program Construction, Lecture Notes in Comput. Sci. 1837, Springer, New York, 2000.
[52] P. Maymin, *Extending the Lambda Calculus to Express Randomized and Quantumized Algorithms*, arXiv:quant-ph/9612052 (1996), http://www.arXiv.org.
[53] P. Maymin, *The Lambda-q Calculus Can Efficiently Simulate Quantum Computers*, arXiv:quant-ph/9702057 (1997), http://www.arXiv.org.
[54] M. Wehr, *Quantum computing: A new paradigm and its type theory*, Presented at Quantum Computing Seminar, Universität Karlsruhe, Germany, 1996.
[55] S. Abramsky and B. Coecke, *Physical traces: Quantum vs. classical information processing*, Electron. Notes Theoret. Comput. Sci., 69, (2003).
[56] E. Kashefi, *Quantum Domain Theory—Definitions and Applications*, arXiv:quant-ph/0306077 (2003), http://www.arXiv.org.
[57] D. Dieks, *Communication by EPR devices*, Phys. Lett. A., 92 (1982), pp. 271–272.
[58] W. K. Wootters and W. H. Zurek, *A single quantum cannot be cloned*, Nature, 299 (1982), pp. 802–803.
[59] H. B. Curry and R. Feys, *Combinatory Logic*, vol. 1, North–Holland, Dordrecht, Netherlands, 1985.
[60] W. A. Howard, *The formulas-as-types notion of construction*, in To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism, J. P. Seldin and J. R. Hindley, eds., Academic Press, New York, 1980, pp. 479–490.
[61] V. Pratt, *Linear logic for generalized quantum mechanics*, in Proceedings of Workshop on Physics and Computation, IEEE, Dallas, 1992, pp. 166–180.
[62] S. Smets, *What has operational quantum logic to do with linear logic?*, presented at Logic and Interaction Week 3, Marseilles, France, 2002.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1136–1170

CIRCUMFERENCE OF GRAPHS WITH BOUNDED DEGREE*

GUANTAO CHEN†, JUN XU‡, AND XINGXING YU§

Abstract. Karger, Motwani, and Ramkumar [*Algorithmica*, 18 (1997), pp. 82–98] have shown that there is no constant approximation algorithm to find a longest cycle in a Hamiltonian graph, and they conjectured that this is the case even for graphs with bounded degree. On the other hand, Feder, Motwani, and Subi [*SIAM J. Comput.*, 31 (2002), pp. 1596–1607] have shown that there is a polynomial time algorithm for finding a cycle of length $n^{\log_3 2}$ in a 3-connected cubic n-vertex graph. In this paper, we show that if G is a 3-connected n-vertex graph with maximum degree at most d, then one can find, in $O(n^3)$ time, a cycle in G of length at least $\Omega(n^{\log_b 2})$, where $b = 2(d-1)^2 + 1$.

Key words. bounded degree, 3-connected components, long cycles and paths, circumference

AMS subject classifications. 05C38, 05C45, 05C85

DOI. 10.1137/S0097539703436473

1. Introduction and notation. The *circumference* of a graph is the length of a longest cycle in that graph. The problem of approximating the circumference of a graph is NP-hard [15]. For many canonical NP-hard problems, either good approximation algorithms have been devised, or strong negative results have been established, leading to better understanding of the approximability of these problems. However, not much is known for finding longest paths and cycles, positive or negative. For example, there is no known algorithm which guarantees an approximation ratio better than $n/\text{polylog}(n)$, where n denotes the number of vertices. This is true even for graphs which are Hamiltonian or have bounded degree. Karger, Motwani, and Ramkumar [15] showed that unless $\mathcal{P} = \mathcal{NP}$, it is impossible to find, in polynomial time, a path of length $n - n^\epsilon$ in an n-vertex Hamiltonian graph for any $\epsilon < 1$. They conjectured that it is just as hard for graphs with bounded degree.

On the positive side, if a graph has a path of length L, then one can find a path of length $\Omega((\log L/\log\log L)^2)$ [1] (also see [20]). Feder, Motwani, and Subi [6] showed that there is a polynomial time algorithm for finding a cycle of length at least $n^{\log_3 2}$ in a 3-connected cubic n-vertex graph. They also showed that if a graph has maximum degree at most three and has a path or cycle of length L, then one can find a path or cycle of length at least $L^{(\log_3^2)/2}$. Therefore, an intermediate problem is to find long paths or cycles in graphs of bounded degree that have a Hamilton cycle. Specifically, Feder, Motwani, and Subi (see [6], p. 1605) asked (1) whether there exists some constant $0 < c < 1$ such that if G is a 3-connected planar n-vertex graph, then the circumference of G is at least $\Omega(n^c)$, and (2) whether there exists some constant $0 < c < 1$ such that if G is a 3-connected n-vertex graph with bounded degree, then

*Received by the editors October 20, 2003; accepted for publication (in revised form) February 17, 2004; published electronically July 20, 2004.
http://www.siam.org/journals/sicomp/33-5/43647.html

†Department of Mathematics and Statistics, Georgia State University, Atlanta, GA 30303. This author was partially supported by NSF grant DMS-0070514 (gchen@mathstat.gsu.edu).

‡College of Computing, Georgia Institute of Technology, Atlanta, GA 30332. This author was partially supported by NSF grant ITR/SY 0113933 and NSF CAREER award ANI 0238315 (jx@cc.gatech.edu).

§Corresponding author. School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332 (yu@math.gatech.edu) and Center for Combinatorics, LPMC, Nankai University, Tianjin 300071, China. This author was partially supported by NSF grant DMS-0245530 and NSA grant MDA904-03-1-0052.

the circumference of G is at least $\Omega(n^c)$. There are known results showing that such a constant c exists in both cases ([3], [14]); however, none addresses the algorithmic issue. The main goal of this paper is to establish a cubic algorithm that produces a long cycle in a 3-connected graph with bounded degree.

The work on circumferences of planar graphs dates back to 1931, when Whitney [21] proved that every 4-connected planar triangulation contains a Hamilton cycle (and, hence, its faces are 4-colorable). This result is generalized to all 4-connected planar graphs in [18]. A linear time algorithm is given in [4] for finding a Hamilton cycle in a 4-connected planar graph. There are many 3-connected planar graphs which do not contain Hamilton cycles (see [9]). On the other hand, the following conjecture of Barnette (see [16]) remains open: every bipartite, cubic, 3-connected, planar graph contains a Hamilton cycle. When studying paths in polytopes, Moon and Moser [17] implicitly conjectured that if G is a 3-connected planar n-vertex graph then G contains a cycle of length at least $\Omega(n^{\log_3 2})$. (Grünbaum and Walther [8] made the same conjecture for a class of 3-connected cubic planar graphs.) Jackson and Wormald [13] gave the first polynomial lower bound $\Omega(n^c)$, where c is approximately 0.2, which was improved to $\Omega(n^{0.4})$ by Gao and Yu [7]. Chung [5] further improved this lower bound to $\Omega(n^{0.5})$. Recently, Chen and Yu [3] fully established the Moon–Moser conjecture; their proof implies a quadratic algorithm for finding a cycle of length at least $\Omega(n^{\log_3 2})$ in a 3-connected planar n-vertex graph. We conjecture that such a cycle may be found in linear time.

The work on circumferences of 3-connected graphs with bounded degree dates back to 1980, when Bondy and Simonovits [2] conjectured that there exists a constant $0 < c < 1$ such that the circumference of any 3-connected cubic n-vertex graph is at least $\Omega(n^c)$. This conjecture was verified by Jackson [12]. In 1993, Jackson and Wormald [14] proved that if G is a 3-connected n-vertex graph with maximum degree at most d, then the circumference of G is at least $\frac{1}{2}n^{\log_b 2} + 1$, where $b = 6d^2$. The argument in [14] is technical, and Jackson and Wormald did not address the algorithmic issue.

In this paper, we improve the lower bound of Jackson and Wormald, for both the exponent and the constant coefficient. Our argument makes efficient use of two results: a convexity result of a function and a decomposition result of 2-connected graphs. Our proof gives rise to a cubic algorithm for finding a long cycle in 3-connected graphs with bounded degree. More precisely, we prove the following result.

THEOREM 1.1. *Let $n \geq 4$ and $d \geq 3$ be integers. Let G be a 3-connected graph on n vertices such that the maximum degree of G is at most d. Then G contains a cycle of length at least $n^{\log_b 2} + 2$, where $b = 2(d-1)^2 + 1$. Moreover, such a cycle can be found in $O(n^3)$ time.*

It is conjectured in [14] that, for $d \geq 4$, the lower bound in Theorem 1.1 may be replaced by $\Omega(n^{\log_{d-1} 2})$. We are hopeful that our approach will eventually lead to a resolution of this conjecture.

To prove Theorem 1.1, we will need to deal with graphs which result from a 3-connected graph by deleting one vertex. Such graphs are 2-connected but not necessarily 3-connected. Our technique is to decompose such a graph into "3-connected components." This can be done in linear time by a result of Hopcroft and Tarjan [10]. (A similar idea is used in [14], but our decomposition is done once for each graph in a single iteration of the algorithm, and we make more efficient use of such a decomposition.) In most situations, we will not use all 3-connected components of a graph. Instead, we will pick some large 3-connected components and find long cycles in such components. We will then use a convexity property of the function $f(x) = x^{\log_b 2}$ to

account for the unused components. These two ideas will be made more precise in the next two sections.

This paper is organized as follows. In section 2, we will state a technical result, consisting of three statements about (a) the existence of a long cycle through a given edge and avoiding a given vertex, (b) the existence of a long cycle through two given edges, and (c) the existence of a long cycle through a given edge. (We will see that (c) implies Theorem 1.1.) We will also describe the decomposition of a 2-connected graph into 3-connected components. In section 3, we will prove useful properties of the convex function $f(x) = x^{\log_b 2}$, for $b = 3$ and $b \geq 4$. We will also prove several lemmas to be used in the proof of our main result. In sections 4–6, we will show that each of (a), (b), and (c) can be reduced in linear time to (a), (b), and/or (c) for smaller graphs. In section 7, we will complete the proof of our main result and give a cubic algorithm that finds a long cycle in a 3-connected graph with bounded degree.

We end this section with notation and terminology to be used throughout this paper. Let G be a graph. We use $V(G)$ and $E(G)$ to denote the vertex set and edge set of G, respectively, and we write $G = (V(G), E(G))$. For convenience, we write $|G|$ instead of $|V(G)|$. If $e \in E(G)$ and x, y are the vertices of G incident with e, then we write $e = xy$. For any $S \subseteq V(G) \cup E(G)$, $G - S$ denotes the graph obtained from G by deleting S and all edges of G with an incident vertex in S. If $S = \{x\}$, then we simply write $G - x$ instead of $G - S$.

Let G and H be two graphs. By $H \subseteq G$ we mean that H is a subgraph of G. We use $G \cup H$ and $G \cap H$ to denote the union and intersection, respectively, of G and H. For any $S \subseteq V(G) \cup E(G)$ and for any $H \subseteq G$, we use $H + S$ to denote the graph with vertex set $V(H) \cup (S \cap V(G))$ and edge set $E(H) \cup \{uv \in S : \{u, v\} \subseteq V(H) \cup (S \cap V(G))\}$.

We say that a graph G is k-connected if $|G| \geq k + 1$ and, for any $S \subseteq V(G)$ with $|S| \leq k - 1$, $G - S$ is connected. Let G be a graph. If $S \subseteq V(G)$ for which $G - S$ is not connected, then S is a *cut* of G, and if, in addition, $|S| = k$, then S is a *k-cut*. If $x \in V(G)$ for which $G - x$ is not connected, then x is called a *cut vertex* of G. If $e \in E(G)$ for which $G - e$ is not connected, then e is called a *cut edge* of G.

2. 3-connected components. We begin this section by stating a technical result which implies Theorem 1.1. To motivate that statement, let G be a 3-connected graph. In order to find a long cycle in G, we will try to find a cycle through a specific edge $e = xy$ (for induction purposes). To reduce the problem to smaller graphs, we consider $G - y$. Clearly $G - y$ is 2-connected but not necessarily 3-connected. In the case when $G - y$ is not 3-connected, y is contained in a 3-cut T of G. Let $T := \{y, a, b\}$, and let G_1, G_2 be subgraphs of G such that $E(G_1) \cap E(G_2) = \emptyset$, $V(G_1) \cap V(G_2) = T$, and $G_1 \cup G_2 = G$. See Figure 1 for an illustration. Assume $x \in V(G_1) - T$. We could find a long cycle C_1 through both e and ab in $G_1 + ab$ and a long cycle C_2 through ab in $(G_2 + ab) - y$, and then $C := (C_1 - ab) \cup (C_2 - ab)$ would give a long cycle in G. Note that C_1 is a cycle through two given edges, C_2 is a cycle through one given edge and avoiding a given vertex, and C is a cycle through one given edge. This suggests that we prove three statements simultaneously. Indeed, we will prove the following.

THEOREM 2.1. *Let $n \geq 5$ and $d \geq 3$ be integers, let $r = \log_{2(d-1)^2+1} 2$, and let G be a 3-connected graph on n vertices. Then the following statements hold:*

(a) *Let $xy \in E(G)$ and $z \in V(G) - \{x, y\}$, and let t denote the number of neighbors of z distinct from x and y. Assume that the maximum degree of G is at most $d + 1$, and every vertex of degree $d + 1$ (if any) is incident with the edge zx or zy. Then there is a cycle C through xy in $G - z$ such that $|C| \geq (\frac{n}{2t})^r + 2$.*

(b) *Suppose the maximum degree of G is at most d. Then, for any distinct $e, f \in E(G)$, there is a cycle C through e and f in G such that $|C| \geq (\frac{n}{2(d-1)})^r + 3$.*

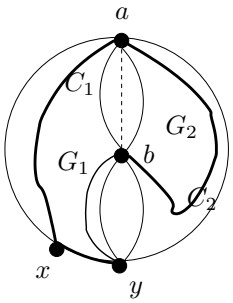


FIG. 1. *An illustration.*

(c) *Suppose that the maximum degree of G is at most d. Then, for any $e \in E(G)$, there is a cycle C through e in G such that $|C| \geq n^r + 3$.*

Clearly, Theorem 2.1(c) implies Theorem 1.1 when $n \geq 5$, and Theorem 1.1 is obvious when $n = 4$. Note the condition in (a) about the maximum degree; it is due to the addition of edges in order to maintain 3-connectivity.

To prove Theorem 2.1, we need to decompose a 2-connected graph (such as $G - z$ in (a) above) into 3-connected components. This is similar to the decomposition of a connected graph into 2-connected components. Let G be a connected graph. A *block* of G is a subgraph of G which is either a maximal 2-connected subgraph of G or a subgraph of G induced by a cut edge of G. A block of G is also called a 2-*connected component of* G. It is easy to see that the intersection of any two blocks of G either is empty or consists of only one vertex (which is a cut vertex). Also any noncut vertex of G occurs in exactly one block of G. This implies that the blocks and cut vertices of G form a tree structure.

Now let G be a 2-connected graph. We describe the 3-connected components of G, following Hopcroft and Tarjan [10]. For this purpose, we allow multiple edges (and hence $E(G)$ is a multiset). We say that $\{a, b\} \subseteq V(G)$ is a *separation pair* in G if there are subgraphs G_1, G_2 of G such that $G_1 \cup G_2 = G$, $V(G_1) \cap V(G_2) = \{a, b\}$, $E(G_1) \cap E(G_2) = \emptyset$, and $|E(G_i)| \geq 2$ for $i = 1, 2$. Let $G'_i := (V(G_i), E(G_i) \cup \{ab\})$ for $i = 1, 2$. See Figure 2 for an example. Then G'_1 and G'_2 are called *split graphs* of G with respect to the separation pair $\{a, b\}$, and the new edge ab added to G_i is called a *virtual* edge. Virtual edges are illustrated with dashed edges in Figures 2–4. It is easy to see that since G is 2-connected, G'_i is 2-connected or G'_i consists of two vertices and at least three multiple edges between them.

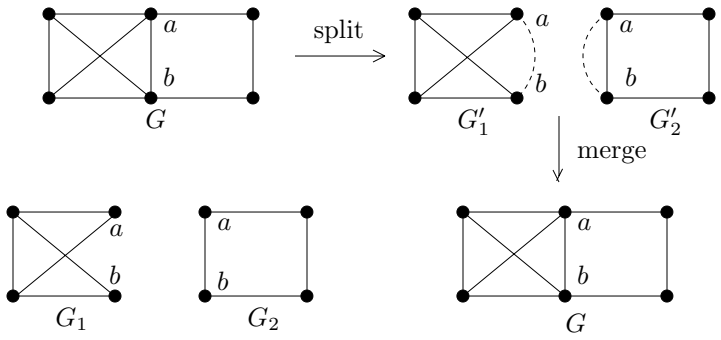


FIG. 2. *Split and merge.*

Suppose that a multigraph is split, and the split graphs are split, and so on, until no more splits are possible. Then each remaining graph is called a *split component*. See Figure 3 for a graph G and its split components. No split component contains a separation pair, and therefore each split component must be one of the following: a triangle, a *triple bond* (two vertices with three multiple edges between), or a 3-connected graph.

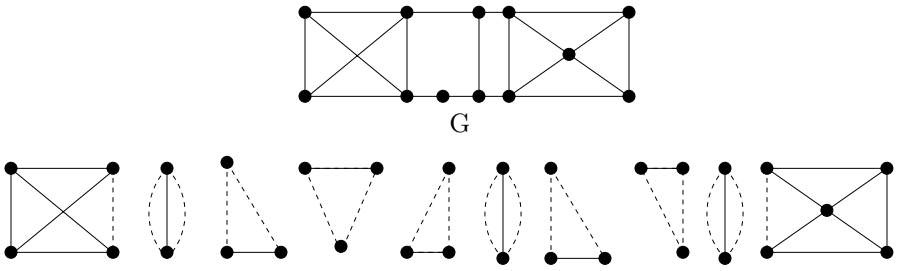


FIG. 3. *Split components of G.*

It is not hard to see that if a split component of a 2-connected graph is 3-connected, then it is unique. It is also easy to see that, for any two split components G_1, G_2 of a 2-connected graph, we have $|V(G_1) \cap V(G_2)| = 0$ or 2, and if $|V(G_1) \cap V(G_2)| = 2$, then either G_1 and G_2 share a virtual edge between vertices in $V(G_1) \cap V(G_2)$ or there is a sequence of triple bonds such that the first shares a virtual edge with G_1, any two consecutive triple bonds in the sequence share a virtual edge, and the last triple bond shares a virtual edge with G_2.

In order to get unique 3-connected components, we need to merge some triple bonds and to merge some triangles. Let $G_i' = (V_i', E_i')$, $i = 1, 2$, be two split components, both containing a virtual edge ab. Let $G' = (V_1' \cup V_2', (E_1' - \{ab\}) \cup (E_2' - \{ab\}))$. Then the graph G' is called the *merge graph* of G_1 and G_2. See Figure 2 for an example of a merge graph. Clearly, a merge of triple bonds gives a graph consisting of two vertices and multiple edges, which is called a *bond*. Also a merge of triangles gives a cycle, and a merge of cycles also gives a cycle.

Let $\mathcal{D}$ denote the set of 3-connected split components of a 2-connected graph G. We merge the other split components of G as follows: the triple bonds are merged as much as possible to give a set of bonds $\mathcal{B}$, and the triangles are merged as much as possible to give a set of cycles $\mathcal{C}$. Then $\mathcal{B} \cup \mathcal{C} \cup \mathcal{D}$ is the set of the 3-*connected components* of G. Figure 4 gives the 3-connected components of the graph in Figure 3. Note that any two 3-connected components either are edge disjoint or share exactly one virtual edge.

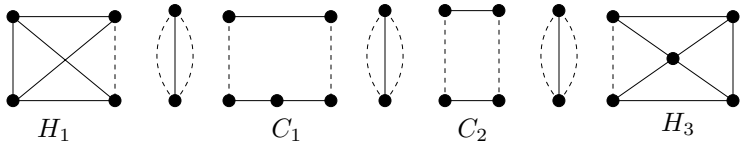


FIG. 4. 3-*connected components of the graph G in Figure* 3.

Tutte [19] proved that the above decomposition of a 2-connected graph into 3-connected components is unique. Hopcroft and Tarjan [10] gave a linear time algorithm for finding the 3-connected components of a graph.

THEOREM 2.2. *For any 2-connected graph, the 3-connected components are unique and can be found in $O(|E|)$ time. Moreover, the total number of edges in the 3-connected components is at most $3|E| - 6$.*

We define a graph whose vertices are the 3-connected components of G, and two vertices are adjacent if the corresponding 3-connected components share a virtual edge. Then it is easy to see that such a graph is a tree, and we call it the *block-bond tree* of G.

For convenience, 3-connected components that are not bonds are called 3-*blocks*. An *extreme* 3-block is a 3-block that contains at most one virtual edge. That is, either it is the only 3-connected component, or it corresponds to a degree one vertex in the block-bond tree.

We will make use of cycle chains. Intuitively, a cycle chain in a 2-connected graph G is a sequence $C_1C_2 \dots C_k$ of 3-blocks of G for which each C_i is a cycle and there exist bonds B_1, B_2, …, B_{k-1} of G such that $C_1B_1C_2B_2 \dots B_{k-1}C_k$ is a path in the block-bond tree of G. More precisely, we have the following.

DEFINITION 2.3. *Let G be a 2-connected graph. By a cycle chain in G we mean a sequence $C_1 \dots C_k$ with the following properties:*

(i) *for each $1 \le i \le k$, C_i is a 3-block of G and C_i is a cycle;*
(ii) *$|V(C_i) \cap V(C_{i+1})| = 2$, and C_i and C_{i+1} each contain a virtual edge between the vertices in $V(C_i) \cap V(C_{i+1})$; and*
(iii) *$|V(C_i) \cap V(C_j)| \le 1$ when $j \ge i + 2$, and if $i < j$ and $|V(C_i) \cap V(C_j)| = 1$, then, for all $i \le t \le j$, $V(C_i) \cap V(C_j) \subseteq V(C_t) \cap V(C_j)$.*

For convenience, we sometimes write $H := C_1 \dots C_k$ and view H as the graph $\bigcup_{i=1}^{k} C_i$. Hence $V(H) := \bigcup_{i=1}^{k} V(C_i)$. Note that H is a multigraph, with two virtual edges between the vertices in $V(C_i) \cap V(C_{i+1})$, $1 \le i \le k-1$.

As an example, take the graph G in Figure 3 and its 3-connected components in Figure 4; we see that C_1C_2 is a cycle chain.

Remark. We choose not to include bonds in our definition of cycle chains because those bonds do not contribute to the vertex count in our arguments.

It is easy to see that if $C_1 \dots C_k$ is a cycle chain, then deleting all virtual edges with both ends in $V(C_i) \cap V(C_{i+1})$, $1 \le i \le k-1$, results in a cycle. We state it as follows.

PROPOSITION 2.4. *Let G be a 2-connected graph, let $C_1 \dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u, v\} \ne V(C_1) \cap V(C_2)$ when $k \ne 1$, and let $xy \in E(C_k)$ with $\{x, y\} \ne V(C_{k-1}) \cap V(C_k)$ when $k \ne 1$. Then $\bigcup_{i=1}^{k} C_i$ contains a Hamilton cycle through uv and xy. Moreover, such a cycle can be found in $O(|\bigcup_{i=1}^{k} V(C_i)|)$ time.*

For later applications, we need several facts about paths in cycle chains. We say that a path P in a graph G is from a vertex $x \in V(G)$ to a set $S \subseteq V(G) - \{x\}$ if one end of P is x, the other end of P is in S, and P is otherwise disjoint from S.

PROPOSITION 2.5. *Let G be a 2-connected graph, let $C_1 \dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u, v\} \ne V(C_1) \cap V(C_2)$ when $k \ne 1$, and let $xy \in E(C_k)$ with $\{x, y\} \ne V(C_{k-1}) \cap V(C_k)$ when $k \ne 1$. Then there is a path in $(\bigcup_{i=1}^{k} C_i) - \{uv, xy\}$ which is from u to $\{x, y\}$ and contains $(\bigcup_{i=1}^{k-1}(V(C_i) \cap V(C_{i+1}))) - (\{x, y\} \cup \{u, v\})$. Moreover, such a path can be found in $O(|\bigcup_{i=1}^{k} V(C_i)|)$ time.*

Proof. If $k = 1$, this is obvious. So assume that $k \ge 2$. Let $x'y'$ denote the virtual edge in C_{k-1} such that $\{x', y'\} = V(C_{k-1}) \cap V(C_k)$. By induction, $(\bigcup_{i=1}^{k-1} C_i) - \{uv, x'y'\}$ contains a path P' that is from u to $\{x', y'\}$ and contains $(\bigcup_{i=1}^{k-2}(V(C_i) \cap V(C_{i+1}))) - (\{x', y'\} \cup \{u, v\})$. By symmetry, we may assume that P' ends at x'.

If $y' \in V(P')$, then let Q' denote the path in $C_k - \{xy, y'\}$ from x' to $\{x, y\}$. If $y' \notin V(P')$, then let Q' denote the path in $C_k - xy$ that is from x' to $\{x, y\}$ and through the virtual edge $x'y'$. Clearly, $P := P' \cup Q'$ gives the desired path.

It is easy to see that such a path can be found in $O(|\bigcup_{i=1}^{k} V(C_i)|)$ time. □

By a similar argument, we can prove the following.

PROPOSITION 2.6. *Let G be a 2-connected graph, let $C_1 \dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u, v\} \neq V(C_1) \cap V(C_2)$ when $k \neq 1$, and let $x \in V(C_k)$ with $x \notin V(C_{k-1})$ when $k \neq 1$. Then there is a path in $(\bigcup_{i=1}^{k} C_i) - uv$ which is from u to x and contains $(\bigcup_{i=1}^{k-1}(V(C_i) \cap V(C_{i+1}))) - \{u, v, x\}$. Moreover, such a path can be found in $O(|\bigcup_{i=1}^{k} V(C_i)|)$ time.*

The next two facts about cycle chains are slightly more complicated. We only prove the first; the other can be proved similarly.

PROPOSITION 2.7. *Let G be a 2-connected graph, let $C_1 \dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u, v\} \neq V(C_1) \cap V(C_2)$ when $k \neq 1$, $ab \in E(C_k)$ with $\{a, b\} \neq V(C_{k-1}) \cap V(C_k)$ when $k \neq 1$, and $cd \in E(\bigcup_{i=1}^{k} C_i) - \{ab\}$. Suppose $ab \neq uv$ when $k = 1$. Then there is a path P in $(\bigcup_{i=1}^{k} C_i) - ab$ from $\{a, b\}$ to $\{c, d\}$ such that $uv \in E(P)$, $cd \notin E(P)$ unless $cd = uv$, and $(\bigcup_{i=1}^{k-1}(V(C_i) \cap V(C_{i+1}))) \subseteq V(P)$. Moreover, such a path can be found in $O(|\bigcup_{i=1}^{k} V(C_i)|)$ time.*

Proof. We apply induction on k. If $k = 1$, then since $ab \neq uv$, the result is obvious. So assume that $k \geq 2$.

First, assume that $cd \in E(C_k)$ and $\{c, d\} \neq V(C_{k-1}) \cap V(C_k)$. Let $a'b'$ denote the virtual edge in C_k with $\{a', b'\} = V(C_{k-1}) \cap V(C_k)$. In $C_k - \{ab, cd\}$, we find a path P' from $\{a, b\}$ to $\{c, d\}$ through $a'b'$. In $\bigcup_{i=1}^{k-1} C_i$, we apply Proposition 2.4 to find a Hamilton cycle C through uv and $a'b'$. Now $P := (P' - a'b') \cup (C - a'b')$ gives the desired path.

Thus we may assume that there is some $1 \leq t < k$ such that $cd \in E(C_t)$. We may choose t so that $\{c, d\} \neq V(C_{t-1}) \cap V(C_t)$ when $t \neq 1$.

Suppose $\{c, d\} = V(C_t) \cap V(C_{t+1})$. By applying Proposition 2.4, we find a Hamilton cycle C in $\bigcup_{i=1}^{t} C_i$ such that $uv, cd \in E(C)$. Now $P' := C - cd$ is a path in $\bigcup_{i=1}^{t} C_i$ from c to d through uv. By Proposition 2.5, we find a path P'' in $(\bigcup_{i=t+1}^{k} C_i) - \{ab, cd\}$ that is from d to $\{a, b\}$ and contains $(\bigcup_{i=t+1}^{k-1}(V(C_i) \cap V(C_{i+1}))) - (\{a, b\} \cup \{c, d\})$. Now $P := P' \cup P''$ gives the desired path.

So assume that $\{c, d\} \neq V(C_t) \cap V(C_{t+1})$. By applying induction, there is a path P' from $V(C_t) \cap V(C_{t+1})$ to $\{c, d\}$ in $\bigcup_{i=1}^{t} C_i$ such that $uv \in E(P')$, $cd \notin E(P')$ unless $cd = uv$, and $(\bigcup_{i=1}^{t-1}(V(C_i) \cap V(C_{i+1}))) \subseteq V(P')$. Let e' denote the virtual edge of C_{t+1} between the vertices in $V(C_t) \cap V(C_{t+1})$, and let $u \in V(C_t) \cap V(C_{t+1})$ be an end of P'. Now apply Proposition 2.5 to $C_{t+1} \dots C_k$, we find a path P'' from u to $\{a, b\}$ in $(\bigcup_{i=t+1}^{k} C_i) - \{e', ab\}$ such that $(\bigcup_{i=t}^{k-1}(V(C_i) \cap V(C_{i+1}))) - (V(C_t) \cap V(C_{t+1})) \subseteq V(P'')$. Clearly, $P := P' \cup P''$ gives the desired path.

Since finding P' and P'' takes $O(|\bigcup_{i=1}^{k} V(C_i)|)$ time, P can also be found in $O(|\bigcup_{i=1}^{k} V(C_i)|)$ time. □

By a similar argument, we can prove the following.

PROPOSITION 2.8. *Let G be a 2-connected graph, let $C_1 \dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u, v\} \neq V(C_1) \cap V(C_2)$ when $k \neq 1$, $x \in V(C_k)$ with $x \notin V(C_{k-1})$ when $k \neq 1$, and $cd \in E(\bigcup_{i=1}^{k} C_i)$. Then there is a path P in $(\bigcup_{i=1}^{k} C_i) - uv$ from x to $\{c, d\}$ such that $uv \in E(P)$, $cd \notin E(P)$ unless $cd = uv$, and $(\bigcup_{i=1}^{k-1}(V(C_i) \cap V(C_{i+1}))) \subseteq V(P)$. Moreover, such a path can be found in $O(|\bigcup_{i=1}^{k} V(C_i)|)$ time.*

We conclude this section by generalizing the concept of a cycle chain to a block chain. Intuitively, a *block chain* in a 2-connected graph G is a sequence $H_1 \dots H_k$ for which (1) each H_i is either a 3-connected 3-block of G or a cycle chain in G and (2) there exist bonds $B_1, \dots, B_{h-1}$ of G such that $H_1 B_1 H_2 B_2 \dots B_{h-1} H_h$ form a path in the block-bond tree of G (by also patching the tree paths corresponding to H_i when H_i is a cycle chain). More precisely, we have the following.

DEFINITION 2.9. *Let G be a 2-connected graph. By a block chain in G we mean a sequence $H_1 \dots H_h$ with the following properties:*

(i) *For each $1 \le i \le h$, either H_i is a 3-connected component of G or H_i is a cycle chain in G, and for $1 \le i \le h-1$, H_i and H_{i+1} cannot both be cycle chains.*

(ii) *For each $1 \le i \le h-1$, $|V(H_i) \cap V(H_{i+1})| = 2$ and both H_i and H_{i+1} contain a virtual edge between the vertices in $V(H_i) \cap V(H_{i+1})$.*

(iii) *$|V(H_i) \cap V(H_j)| \le 1$ if $3 \le i+2 \le j \le h$, and if $1 \le i < j \le h$ and $|V(H_i) \cap V(H_j)| = 1$, then, for all $i \le t \le j$, $V(H_i) \cap V(H_j) \subseteq V(H_t) \cap V(H_j)$.*

(iv) *Suppose $H_i = C_1 C_2 \dots C_k$ is a cycle chain. If $i < h$, then $V(H_{i+1}) \cap V(H_i) \subseteq V(C_k)$ and $V(H_{i+1}) \cap V(H_i) \ne V(C_{k-1}) \cap V(C_k)$, and if $i > 1$, then $V(H_{i-1}) \cap V(H_i) \subseteq V(C_1)$ and $V(H_{i-1}) \cap V(H_i) \ne V(C_1) \cap V(C_2)$.*

For convenience, we denote $\mathcal{H} = H_1 \dots H_h$ and view $\mathcal{H}$ as the graph $\bigcup_{i=1}^{h} H_i$. Thus $V(\mathcal{H}) := \bigcup_{i=1}^{h} V(H_i)$. Note that $\mathcal{H}$ is a multigraph, with two virtual edges between vertices in $V(H_i) \cap V(H_{i+1})$, $1 \le i \le h-1$.

In Figure 4, $\mathcal{H} = H_1 H_2 H_3$ is a block chain in G, where H_2 is the cycle chain $C_1 C_2$. In a block chain, we do not include bonds, because bonds do not contribute to the vertex count in our arguments.

3. Technical lemmas. In this section we prove several lemmas to be used in the proof of Theorem 2.1. Notice $G = G_1 \cup G_2$ in the illustration of Figure 1. If, instead, for some $k \ge 3$, $G = \bigcup_{i=1}^{k} G_i$, $E(G_i) \cap E(G_j) = \emptyset$, and $|V(G_i) \cap V(G_j)| = 3$ for $1 \le i < j \le k$, then the following lemma will enable us to conclude that if $|G_1|$ and $|G_2|$ are the largest among all $|G_i|$'s, then the cycle C produced by finding long cycles C_i in G_i (as in the first paragraph in section 2), $i = 1, 2$, will be long as well.

LEMMA 3.1. *Let $b = 3$ or $b \ge 4$ be an integer, and let m, n be positive integers with $m \ge n$. Then $m^{\log_b 2} + n^{\log_b 2} \ge (m + (b-1)n)^{\log_b 2}$.*

Proof. By dividing both sides of the above inequality by $m^{\log_b 2}$, it suffices to show that, for any s with $0 \le s \le 1$,

$$1 + s^{\log_b 2} \ge (1 + (b-1)s)^{\log_b 2}.$$

Let $f(s) = 1 + s^{\log_b 2} - (1 + (b-1)s)^{\log_b 2}$. Clearly, $f(0) = f(1) = 0$. Taking the derivative about s, we have

$$f'(s) = (\log_b 2)(s^{(\log_b 2)-1} - (b-1)(1 + (b-1)s)^{(\log_b 2)-1}).$$

A simple calculation shows that $f'(s) = 0$ has a unique solution. Therefore, since $f(0) = f(1) = 0$, either 0 is the absolute maximum of $f(s)$ over $[0, 1]$ or 0 is the absolute minimum of $f(s)$ over $[0, 1]$. That is, either $f(s) \ge 0$ for all $s \in [0, 1]$ or $f(s) \le 0$ for all $s \in [0, 1]$. Note that $0 < \frac{1}{b} < 1$ (since $b \ge 3$) and

$$\begin{aligned} f\left(\frac{1}{b}\right) &= \left(1 + \frac{1}{2}\right) - \left(1 + \frac{b-1}{b}\right)^{\log_b 2} \\ &= \frac{3}{2} - \frac{(2b-1)^{\log_b 2}}{2}. \end{aligned}$$

We claim that $f(\frac{1}{b}) > 0$. If $b = 3$, then $f(\frac{1}{b}) = \frac{1}{2}(3 - 5^{\log_3 2}) > 0$. So assume $b \geq 4$. Then $f(\frac{1}{b}) > \frac{3}{2} - \frac{(2b)^{\log_b 2}}{2} = \frac{3}{2} - 2^{\log_b 2}$. Since $b \geq 4$, $2^{\log_b 2} \leq 2^{\log_4 2} = \sqrt{2} < \frac{3}{2}$. Thus $f(\frac{1}{b}) > 0$ for $b \geq 4$.

Therefore, we have $f(s) \geq 0$ for all $s \in [0, 1]$. □

We remark that Lemma 3.1 holds for $b \geq 3$. We choose to state it for $b = 3$ and $b \geq 4$ for simplicity in calculations.

The observations in the following lemma will be convenient in the proof of Theorem 2.1.

LEMMA 3.2. *Let m be an integer, $b \geq 4$, and $d \geq 3$. If $m \geq 4$, then $m \geq m^{\log_b 2}+2$. If $m \geq 3$, then $m > (\frac{m}{2(d-1)})^{\log_b 2}+2$. If $m \geq 2$, then $m > (\frac{m}{2(d-1)})^{\log_b 2}+1$.*

Proof. Let $f(x) = x - x^{\log_b 2}$. We can show that $f'(x) > 0$ for $x \geq 4$. Hence $f(x)$ is an increasing function when $x \geq 4$. Thus, $f(x) \geq f(4) = 4 - 4^{\log_b 2} \geq 2$ (since $b \geq 4$). Thus when $m \geq 4$, we have $m \geq m^{\log_b 2} + 2$.

Next, let $f(x) = x - (\frac{x}{2(d-1)})^{\log_b 2}$. We can show that $f(x)$ is an increasing function when $x \geq 2$. The second inequality follows from $f(x) \geq f(3) > 2$, and the third inequality follows from $f(x) \geq f(2) > 1$. □

After we decompose a 2-connected graph into 3-connected components, we need to find long cycles in certain 3-connected components. This will be done by inductively applying (a), (b), or (c) of Theorem 2.1 to 3-connected components or to graphs obtained from 3-connected components by an "H-transform" or "T-transform."

Let G be a graph and let e, f be distinct edges of G. An *H-transform* of G *at* $\{e, f\}$ is an operation that subdivides e and f by vertices x and y, respectively, and then adds the edge xy. See Figure 5. Let G be a graph, let $e \in E(G)$, and let $x \in V(G)$, which is not incident with e. A *T-transform* of G *at* $\{x, e\}$ is an operation that subdivides e with a vertex y and then adds the edge xy. If there is no need to specify e, f, x, we will simply speak of an H-transform or a T-transform. The following result is easy to prove.

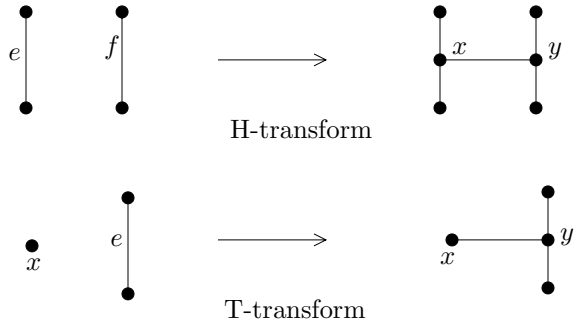


FIG. 5. *H-transform and T-transform.*

LEMMA 3.3. *Let $d \geq 3$ be an integer, and let G be a 3-connected graph with maximum degree at most d. Let G' be a graph obtained from G by an H-transform or a T-transform. Then G' is 3-connected graph, the vertex of G involved in the T-transform has degree at most $d + 1$, and all other vertices of G' have degree at most d.*

Next, we state two results from [11]. The first says that any k-connected graph contains a sparse k-connected spanning subgraph.

LEMMA 3.4. *Let G be a k-connected graph, where k is a positive integer. Then*

G contains a k-connected spanning subgraph with $O(|G|)$ edges, and such a subgraph can be found in $O(|E(G)|)$ time.

The next result is an easy consequence of a result in [11], which states that, in a 2-connected graph G, one can find, in $O(|G|)$ time, two disjoint paths between two given vertices.

LEMMA 3.5. *Let G be a 2-connected graph and let $e, f \in E(G)$. Then there is a cycle through e and f in G, and such a cycle can be found in $O(|G|)$ time.*

The final two results of this section deal with the existence of certain paths in a 3-connected graph. Since such paths need to be produced (when finding a long cycle), we also show that they can be found in linear time. The proofs of these two results are almost identical, so we omit the details of the second proof.

LEMMA 3.6. *Let G be a 3-connected graph, let $f \in E(G)$, let $ab, cd, vw \in E(G) - \{f\}$, and assume that $\{c, d\} \neq \{v, w\}$. Then there exists a path P from $\{a, b\}$ to some $z \in \{c, d\} \cup \{v, w\}$ in G such that*

(i) *$f \in E(P)$,*
(ii) *$cd \in E(P)$ or $vw \in E(P)$, and*
(iii) *if $cd \in E(P)$, then $z \in \{v, w\}$ and $vw \notin E(P)$, and if $vw \in E(P)$, then $z \in \{c, d\}$ and $cd \notin E(P)$.*

Moreover, such a path can be found in $O(|G|)$ time.

Note that in (iii) above when $vw \notin E(P)$, it is possible that $v \in V(P)$ and/or $w \in V(P)$.

Proof. First, we find a cycle C through both ab and f. This can be done in $O(|G|)$ time using Lemma 3.5. Next we distinguish three cases. Note that checking these cases can be done in $O(|G|)$ time.

Case 1. $cd, vw \in E(C)$. In this case one of the following holds: f and vw are contained in a component P of $C - \{ab, cd\}$, or f and cd are contained in a component P of $C - \{ab, vw\}$. In either case, P gives the desired path and can be found in $O(|G|)$ time.

Case 2. $cd \notin E(C)$ and $vw \in E(C)$, or $cd \in E(C)$ and $vw \notin E(C)$. By symmetry, assume that $cd \notin E(C)$ and $vw \in E(C)$. Let Q_1 and Q_2 denote the components of $C - \{ab, f\}$, and assume that $vw \in E(Q_1)$. By Lemma 3.5, we can find, in $O(|G|)$ time, disjoint paths P_1 and P_2 from c, d to some vertices c', d', respectively, of C which are also disjoint from $C - \{c', d'\}$.

If $c' \in V(Q_2)$ or $d' \in V(Q_2)$, then $(C \cup P_1 \cup P_2) - \{ab, cd\}$ contains a path P from $\{a, b\}$ to $\{c, d\}$ through f and vw. So assume that $c', d' \in V(Q_1)$.

If vw is contained in the subpath of Q_1 between c' and d', then $(C \cup P_1 \cup P_2) - \{ab, vw\}$ contains a path P from $\{a, b\}$ to $\{v, w\}$ through f and cd.

If $\{c', d'\}$ is contained in the subpath of Q_1 between vw and ab, then $(C \cup P_1 \cup P_2) - \{ab, cd\}$ contains a path P from $\{a, b\}$ to $\{c, d\}$ through f and vw.

Assume that $\{c', d'\}$ is contained in the subpath of Q_1 between vw and f. Then $(C \cup P_1 \cup P_2) - \{ab, vw\}$ contains a path P from $\{a, b\}$ to $\{v, w\}$ through f and cd.

Note that the above cases can be checked in constant time, and in each case, P can be found in $O(|G|)$ time.

Case 3. $cd \notin E(C)$ and $vw \notin E(C)$. Let Q_1 and Q_2 denote the components of $C - \{ab, f\}$. By Lemma 3.5, there are disjoint paths P_1 and P_2 in G from c, d to $c', d' \in V(C)$, respectively, which are also disjoint from $C - \{c', d'\}$ (and can be found in $O(|G|)$ time). We may assume that $\{c', d'\} \not\subseteq V(Q_i)$ for $i = 1, 2$; otherwise, $C \cup P_1 \cup P_2$ contains a cycle through ab, cd, f and, as in Cases 1 and 2, we can find the desired path in $O(|G|)$ time. Thus, by symmetry, we may assume that $c' \in V(Q_1)$

and $d' \in V(Q_2)$.

If $vw \in E(P_1 \cup P_2)$, then $(C \cup P_1 \cup P_2) - \{ab, vw\}$ contains a path P from $\{a, b\}$ to $\{v, w\}$ through f and cd, which can be found in $O(|G|)$ time. So assume that $vw \notin E(P_1 \cup P_2)$. Therefore, by Lemma 3.5, we can find, in $O(|G|)$ time, disjoint paths R_1, R_2 from v, w to $v', w' \in V(C \cup P_1 \cup P_2)$, respectively, which are also disjoint from $(C \cup P_1 \cup P_2) - \{v', w'\}$.

By similar arguments, we may assume that $\{v', w'\} \not\subseteq V(Q_i)$ (or we go back to Case 1 or Case 2) and $\{v', w'\} \not\subseteq V(P_i)$ for any $i \in \{1, 2\}$ (or we could have chosen P_1, P_2 to include vw and have gone back to the case in the previous paragraph).

Subcase 3.1. $\{v', w'\} \neq \{c', d'\}$. First assume $\{v', w'\} \subseteq V(P_1 \cup P_2)$. Then cd belongs to the subpath of $(P_1 \cup P_2) + cd$ between v' and w'. We see that there is a path P in $((C - ab) \cup P_1 \cup P_2 \cup R_1 \cup R_2) + cd \subseteq G - \{ab, vw\}$ from $\{a, b\}$ to $\{v, w\}$ through both f and cd.

Now assume $\{v', w'\} \subseteq V(Q_1 \cup Q_2)$. Then by symmetry, we may assume that $v' \in V(Q_1)$, $w' \in V(Q_2)$, and $w' \neq d'$. If f, w', d', ab occur on C in cyclic order, then there is a path P from $\{a, b\}$ to $\{v, w\}$ through f and cd in $((C - ab) \cup P_1 \cup P_2 \cup R_1 \cup R_2) + cd \subseteq G - \{ab, vw\}$. If f, d', w', ab occur on C in cyclic order, then there is a path P from $\{a, b\}$ to $\{c, d\}$ through both f and vw in $((C - ab) \cup P_1 \cup P_2 \cup R_1 \cup R_2) + vw \subseteq G - \{ab, cd\}$.

Thus we may assume by symmetry that $v' \in (V(P_1) \cup V(P_2)) - \{c', d'\}$ and $w' \in (V(Q_1) \cup V(Q_2)) - \{c', d'\}$. It is easy to see that $((C - ab) \cup P_1 \cup P_2 \cup R_1 \cup R_2) + vw \subseteq G - \{ab, cd\}$ contains a path P from $\{a, b\}$ to $\{c, d\}$ through both f and vw.

The above three cases can be checked in $O(|G|)$ time, and in all cases, P can be found in $O(|G|)$ time.

Subcase 3.2. $\{v', w'\} = \{c', d'\}$. Let S_1 and S_2 denote the paths between c' and d' in C containing f and ab, respectively. Since G is 3-connected, there is a path S from some $s \in V(R_1 \cup R_2 \cup S_2) - \{c', d'\}$ to $s' \in V(P_1 \cup P_2 \cup S_1) - \{c', d'\}$, which is also disjoint from $(C \cup P_1 \cup P_2 \cup R_1 \cup R_2) - \{s, s'\}$. Note that S can be found in $O(|G|)$ time.

If $s \in V(S_2)$, then $((C - ab) \cup P_1 \cup P_2 \cup R_1 \cup R_2 \cup S) + cd \subseteq G - \{ab, vw\}$ contains a path P from $\{a, b\}$ to $\{v, w\}$ through both f and cd. If $s' \in V(S_1)$, then $((C - ab) \cup P_1 \cup P_2 \cup R_1 \cup R_2 \cup S) + vw \subseteq G - \{ab, cd\}$ contains a path P from $\{a, b\}$ to $\{c, d\}$ through both f and vw, or $((C - ab) \cup P_1 \cup P_2 \cup R_1 \cup R_2 \cup S) + cd \subseteq G - \{ab, vw\}$ contains a path P from $\{a, b\}$ to $\{v, w\}$ through both f and cd.

Therefore, we may assume that $s \in V(R_1 \cup R_2) - \{c', d'\}$ and $s' \in V(P_1 \cup P_2) - \{v', w'\}$. Then $((C - ab) \cup P_1 \cup P_2 \cup R_1 \cup R_2 \cup S) + vw \subseteq G - \{ab, cd\}$ contains a path P from $\{a, b\}$ to $\{c, d\}$ through both f and vw.

The above three cases can be checked in constant time, and in all cases, P can be found in $O(|G|)$ time. □

LEMMA 3.7. *Let G be a 3-connected graph, let $f \in E(G)$, let $x \in V(G)$ which is not incident with f, let $cd, vw \in E(G) - \{f\}$, and assume that $\{c, d\} \neq \{v, w\}$. Then there exists a path P in G from x to some $z \in \{c, d\} \cup \{v, w\}$ such that*

(i) *$f \in E(P)$,*
(ii) *$cd \in E(P)$ or $vw \in E(P)$, and*
(iii) *if $cd \in E(P)$ then $z \in \{v, w\}$ and $vw \notin E(P)$, and if $vw \in E(P)$, then $z \in \{c, d\}$ and $cd \notin E(P)$.*

Moreover, such a path can be found in $O(|G|)$ time.

Proof. The proof is the same as for Lemma 3.6, with ab replaced by x, and when finding paths P_i, R_i, we apply Lemma 3.5 to $G - x$ (which is 2-connected). □

4. Cycles avoiding a vertex. In this section, we show how to reduce Theorem 2.1(a) to (b) and/or (c) of the same theorem in linear time. First, we state the reduction as a lemma.

LEMMA 4.1. *Let $n \geq 6$ and $d \geq 3$ be integers, let $r = \log_{2(d-1)^2+1} 2$, and assume that Theorem 2.1 holds for graphs with at most $n-1$ vertices. Let G be a 3-connected graph with n vertices, let $xy \in E(G)$ and $z \in V(G) - \{x, y\}$, and let t denote the number of neighbors of z distinct from x and y. Assume that the maximum degree of G is at most $d+1$, and every vertex of degree $d+1$ in G (if any) is incident with the edge zx or zy. Then there is a cycle C through xy in $G-z$ such that $|C| \geq (\frac{n}{2t})^r + 2$.*

Proof. We consider $G - z$. Since the vertices of G with degree $d+1$ must be incident with the edge yz or xz, the maximum degree of $G - z$ is at most d.

First, assume that $G - z$ is 3-connected. Since $n \geq 6$, $|G - z| \geq 5$, and hence Theorem 2.1 holds for $G - z$. By Theorem 2.1(c), $G - z$ contains a cycle C through xy such that

$$\begin{aligned} |C| &\geq (n-1)^r + 3 \\ &= ((n-1)^r + 1) + 2 \\ &\geq ((n-1)+1)^r + 2 \quad \text{(by Lemma 3.1)} \\ &> \left(\frac{n}{2t}\right)^r + 2. \end{aligned}$$

Therefore, we may assume that $G - z$ is not 3-connected. By Theorem 2.2, we can decompose $G - z$ into 3-connected components. Let $\mathcal{H} = H_1 \dots H_h$ be a block chain in $G - z$ such that

(i) $\{x, y\} \subseteq V(H_1)$, and $\{x, y\} \neq V(H_1) \cap V(H_2)$ when $k \neq 1$,
(ii) H_h contains an extreme 3-block of $G - z$, and
(iii) subject to (i) and (ii), $|V(\mathcal{H})|$ is maximum.

Note that $H_1 \dots H_k$ can be found in $O(|G|)$ time.

We claim that $|V(\mathcal{H})| \geq \frac{n-1}{t}$. Since G is 3-connected, each extreme 3-block of $G-z$ distinct from H_1 contains a neighbor of z that is not incident with xy. Therefore, there are at most t extreme 3-blocks of $G - z$ different from H_1. Thus there are at most t different block chains in $G-z$ starting with a 3-block or cycle chain containing $\{x, y\}$ and ending with an extreme 3-block of $G-z$ or a cycle chain in $G-z$ containing an extreme 3-block. Since all such chains cover the whole graph $G-z$, it follows from (iii) that $|V(\mathcal{H})| - 2 \geq \frac{n-3}{t}$, and thus $|V(\mathcal{H})| \geq \frac{n-1}{t}$.

Let $V(H_i) \cap V(H_{i+1}) = \{x_i, y_i\}$, $1 \leq i \leq h-1$, and assume that the notation is chosen so that H_i contains disjoint paths from x_{i-1}, y_{i-1} to x_i, y_i, respectively, where $x_0 = x$ and $y_0 = y$. See Figure 6. Next, we show how to find the desired cycle in $G - z$.

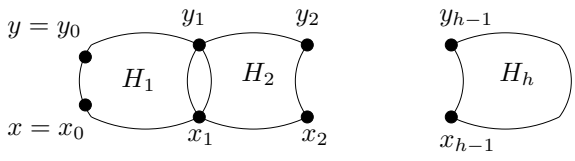


FIG. 6. *Block chain $\mathcal{H} = H_1 \dots H_h$.*

Case 1. There exists some $1 \leq i \leq h$ such that $|H_i| \geq \frac{n}{2t}$. We choose H_i so that $|H_i| \geq |H_j|$ for all $1 \leq j \leq h$. Then $|H_i| \geq \frac{n}{2t}$.

First, assume that $|H_i| = 3$. Then $3 \geq \frac{n}{2t}$. By the choice of H_i, $|H_j| = 3$ for all

$1 \le j \le h$. Since $\mathcal{H}$ does not contain two consecutive cycle chains, we have $h = 1$. Hence $G - z$ is a union of triangles which share the edge xy. Therefore, there are exactly t triangles. Because $n \ge 6$, we have $t \ge 3$, and thus $n = t + 3 \le 2t$. Hence $C := H_i$ is a cycle through xy in $G - z$, and $|C| = 3 \ge (\frac{n}{2t})^r + 2$.

We may assume that $|H_i| \ge 4$. If $H_i = K_4$ or H_i is a cycle chain in $G - z$, then, by Proposition 2.4, let C_i denote a Hamilton cycle through $x_{i-1}y_{i-1}, x_iy_i$ in H_i. By Lemma 3.2, $|C_i| = |H_i| \ge (|H_i|)^r + 2 \ge (\frac{n}{2t})^r + 2$.

Thus assume that H_i is 3-connected and $|H_i| \ge 5$. Since $|H_i| < |G|$ and the maximum degree of H_i is at most d, Theorem 2.1 holds for H_i. By Theorem 2.1 (c), there is a cycle C_i through $e_i := x_{i-1}y_{i-1}$ in H_i such that $|C_i| \ge |H_i|^r + 3 \ge (\frac{n}{2t})^r + 2$.

We can obtain a cycle C in G by replacing virtual edges contained in C_i with paths in G (in particular, replacing $x_{i-1}y_{i-1}$ by a path through xy in G). Therefore, C is a cycle through xy in G, and $|C| \ge |C_i| \ge (\frac{n}{2t})^r + 2$.

Case 2. For each $1 \le i \le h$, $|H_i| < \frac{n}{2t}$. Since $|V(\mathcal{H})| \ge \frac{n-1}{t} > \frac{n}{2t}$, we have $h \ge 2$. We will find a cycle C_i through $x_{i-1}y_{i-1}$ and x_iy_i in each H_i, where x_hy_h is an arbitrary edge of H_h.

If $H_i = K_4$ or H_i is a cycle chain in $G - z$, then, by Proposition 2.4, let C_i be a Hamilton cycle through both $x_{i-1}y_{i-1}$ and x_iy_i in H_i. By Lemma 3.2, $|C_i| = |H_i| \ge (\frac{|H_i|}{2(d-1)})^r + 2$.

Assume that $H_i \ne K_4$ and H_i is not a cycle chain in $G - z$. Then $|H_i| \ge 5$ and H_i is a 3-connected graph with maximum degree d. Since $|H_i| < n$, Theorem 2.1 holds for H_i. By Theorem 2.1 (b), there is a cycle C_i through $x_{i-1}y_{i-1}$ and x_iy_i in H_i such that $|C_i| \ge (\frac{|H_i|}{2(d-1)})^r + 3$.

Note that $C_1 - x_1y_1$, $C_h - x_{h-1}y_{h-1}$, and $C_i - \{x_{i-1}y_{i-1}, x_iy_i\}$, $2 \le i \le h-1$, are edge disjoint, and their union is a cycle C' through xy in $\mathcal{H}$. By replacing the virtual edges in C' with paths in G, we can produce a cycle C through xy in $G - z$ such that $|C| \ge |C'|$. Hence $|C| \ge (\frac{|H_1|}{2(d-1)})^r + \cdots + (\frac{|H_h|}{2(d-1)})^r + 2$.

Note that $h \ge 2$ and the vertices in $V(H_1) \cap V(H_2)$ are counted twice in $|H_1| + \cdots + |H_h|$. Hence $|H_1| + \cdots + |H_h| > \frac{n-1}{t} + 1 \ge \frac{n}{t}$. Consider the function $f(x_1, \ldots, x_h) = x_1^r + \cdots + x_h^r + 2$, with $x_1 + \cdots + x_h \ge \frac{n}{2(d-1)t}$ and $0 \le x_i \le \frac{n}{4(d-1)t}$. By the convexity of $f(x_1, \ldots, x_h)$, the minimum of $f(x_1, \cdots, x_h)$ is achieved on the boundary of its domain. In particular, the minimum is achieved when $x_1 = x_2 = \frac{n}{4(d-1)t}$ and $x_3 = \cdots = x_h = 0$. Hence

$$\begin{aligned} f(x_1, \ldots, x_h) &\ge f\left(\frac{n}{4(d-1)t}, \frac{n}{4(d-1)t}, 0, \ldots, 0\right) \\ &= 2\left(\frac{n}{4(d-1)t}\right)^r + 2 \\ &> \left(\frac{n}{2t}\right)^r + 2. \end{aligned}$$

The final inequality follows from the fact that $r = \log_{2(d-1)^2+1} 2$ and $2 = (2(d-1)^2 + 1)^r$. Therefore, $|C| \ge (\frac{n}{2t})^r + 2$. □

As we can see from the above proof, the desired cycle through xy in $G - z$ can be found either (1) directly, (2) by finding a long cycle through e_i in some H_i with $|H_i| \ge \frac{n}{2t}$, or (3) by finding long cycles through $x_{j-1}y_{j-1}$ and x_jy_j in H_j, $1 \le j \le h$. Next we show that the proof of Lemma 4.1 implies that this process can be done in $O(|E(G)|)$ time.

Algorithm Avoidvertex. Let G be a 3-connected graph, $e = xy \in E(G)$, and

$z \in V(G) - \{x, y\}$, satisfying the conditions of Lemma 4.1. The algorithm performs the following steps.

1. *Preprocessing.* Replace G by a 3-connected spanning graph of G with $O(|G|)$ edges. (This can be done in $O(|E(G)|)$ time using Lemma 3.4.)
2. Decompose $G - z$ into 3-connected components. (This can be done in $O(|G|)$ time using Lemma 2.2.)
3. If there is only one 3-block of $G - z$, then $G - z$ is 3-connected and we proceed to find a cycle D through $e = xy$ in $G - z$ such that $|D| \geq (|G|-1)^r + 3$. That is, we reduce (a) for G, xy, z to (c) for $G - z, xy$. (Clearly, this reduction can be done in constant time.)
4. If there are at least two 3-blocks of $G - z$, then $G - z$ is not 3-connected. We find a block chain $\mathcal{H} = H_1 \dots H_k$ in $G - z$ such that $\{x, y\} \subseteq V(H_1)$, $\{x, y\} \neq V(H_1) \cap V(H_2)$, and $|\mathcal{H}| \geq \frac{n-1}{t}$. Let $V(H_i) \cap V(H_{i+1}) = \{x_i, y_i\}$ for $1 \leq i \leq h-1$. (Note that $\mathcal{H}$ can be found in $O(|G|)$ time by a simple search.)
5. Either find some H_i with $|H_i| \geq \frac{n}{2t}$, or certify that $|H_i| < \frac{n}{2t}$ for all $1 \leq i \leq h$. (This can be done in $O(|G|)$ time by a simple search.)
6. Suppose there exists some $1 \leq i \leq h$ for which $|H_i| \geq \frac{n}{2t}$.
   - If $H_i = K_4$ or H_i is a cycle chain, then let C_i denote a Hamilton cycle in H_i through the edge $x_{i-1}y_{i-1}$. Let C be a cycle in G obtained from C_i by replacing virtual edges with paths in G, and make sure $e \in E(C)$. (Note that C_i can be found in $O(|H_i|)$ time using Proposition 2.4, and so C can be found in $O(|G|)$ time.)
   - If H_i is 3-connected and $H_i \neq K_4$, then to find the desired cycle in $G - z$ through e it suffices to find a cycle D in H_i through $x_{i-1}y_{i-1}$ such that $|D| \geq |H_i|^r + 3$. Hence, we reduce (a) for G, e, z to (c) for $H_i, x_{i-1}y_{i-1}$. (This can be done in constant time.)
7. Now assume that, for all $1 \leq j \leq h$, $|H_j| < \frac{n}{2t}$. Then $h \geq 2$. For each $1 \leq j \leq h$, we perform the following:
   - If $H_j = K_4$ or H_j is a cycle chain in $G - z$, let C_j denote a Hamilton cycle through both $x_{j-1}y_{j-1}$ and x_jy_j in H_j. (Note that C_j can be found in $O(|H_j|)$ time using Proposition 2.4.)
   - If H_j is 3-connected and $H_j \neq K_4$, then it suffices to find a cycle D in H_j through $x_{j-1}y_{j-1}$ and x_jy_j such that $|D| \geq (\frac{|H_j|}{2(d-1)})^r + 3$. Hence, we reduce (a) for G, xy, z to (b) for $H_j, x_{j-1}y_{j-1}, x_jy_j$, for all H_j which are not cycle chains and are not isomorphic to K_4. (Clearly, this can done in $O(|G|)$ time. Moreover, any such H_j contains a vertex that does not belong to any other H_k—this is why we want $H_j \neq K_4$, and it will be used in the final complexity analysis.)

The correctness of the algorithm follows from the proof of Lemma 4.1. To summarize, we have the following result.

PROPOSITION 4.2. *Let $G, e = xy, z, t, d, r$ be as in Theorem* 2.1(a). *Then, in $O(|E(G)|)$ time, we can either*

(1) *find a cycle C through e in $G - z$ with $|C| \geq (\frac{|G|}{2t})^r + 2$,*
(2) *reduce* (a) *of Theorem* 2.1 *for G, xy, z to* (c) *of Theorem* 2.1 *for some* 3*-block H_i of $G - z$ that is* 3*-connected and $|H_i| \geq \max\{5, \frac{|G|}{2t}\}$, or*
(3) *reduce* (a) *of Theorem* 2.1 *for G, xy, z to* (b) *of Theorem* 2.1 *for $H_j, x_{j-1}y_{j-1}$, x_jy_j for some* 3*-connected* 3*-blocks $H_j \neq K_4$.*

Moreover, in (3)*, each H_j contains a vertex that does not belong to any other H_k, $k \neq j$.*

5. Cycles through two edges. In this section, we show how to reduce (b) of Theorem 2.1 to (a) or (b) of the same theorem for smaller graphs. We will show that such a reduction can be performed in linear time.

LEMMA 5.1. *Let $n \geq 6$ and $d \geq 3$ be integers, let $r = \log_{2(d-1)^2+1} 2$, and assume that Theorem 2.1 holds for graphs with at most $n-1$ vertices. Suppose G is a 3-connected graph on n vertices and that the maximum degree of G is at most d. Then for any $\{e, f\} \subseteq E(G)$ there is a cycle C through e, f in G such that $|C| \geq (\frac{n}{2(d-1)})^r + 3$.*

Proof. First, assume that e is incident with f. Let $e = xz$ and $f = yz$, and let $G' := G + xy$. Then G' is a 3-connected graph with maximum degree at most $d+1$, and the possible vertices of degree $d+1$ in G' are x and y, which are incident with the edge zx or zy. By applying Lemma 4.1 to G', xy, z, there is a cycle C' through xy in $G' - z$ such that $|C'| \geq (\frac{n}{2t})^r + 2$, where t is the number of neighbors of z in G' distinct from x and y. Since $zx, zy \in E(G)$, $t \leq d-1$. Now let $C := C' - xy + \{e, f\}$. Then $|C| \geq (\frac{n}{2t})^r + 3 \geq (\frac{n}{2(d-1)})^r + 3$, and C gives the desired cycle.

Therefore, we may assume that e and f are not incident. Let $e = xy$, and consider $G - y$. Since y is not incident with f, $f \in E(G - y)$. Since G is 3-connected, $G - y$ is 2-connected.

Suppose that $G - y$ is 3-connected. Let $y' \neq x$ be a neighbor of y. Then $G' := (G-y)+xy'$ is a 3-connected graph with maximum degree at most d, and $5 \leq |G'| < n$. Hence Theorem 2.1 holds for G'. By Theorem 2.1(b), there is a cycle C' through xy' and f in G' such that $|C'| \geq (\frac{n-1}{2(d-1)})^r + 3$. Let $C := (C' - xy') + \{y, xy, yy'\}$. Then

$$\begin{aligned} |C| &= |C'| + 1 \\ &\geq \left(\frac{n-1}{2(d-1)}\right)^r + 3 + 1 \\ &\geq \left(\frac{n-1}{2(d-1)} + 1\right)^r + 3 \quad \text{(by Lemma 3.1)} \\ &> \left(\frac{n}{2(d-1)}\right)^r + 3. \end{aligned}$$

Assume that $G - y$ is not 3-connected. By Theorem 2.2, we can decompose $G - y$ into 3-connected components. Let $\mathcal{H} := H_1 \dots H_h$ be a block chain in $G - y$ such that $x \in V(H_h)$, $x \notin V(H_{h-1})$ when $h \geq 2$, $f \in E(H_1)$, and f is not incident with both vertices in $V(H_1) \cap V(H_2)$. See Figure 7. Define $V(H_s) \cap V(H_{s+1}) = \{a_s, b_s\}$ for $1 \leq s \leq h-1$.

For each $1 \leq s \leq h$ we define A_s, which consists of vertices of H_s to be counted when applying induction. If H_s is 3-connected, then let $A_s := V(H_s)$. If $h = 1$ and $H_s = C_1 \dots C_k$ is a cycle chain in $G - y$, then let A_s consist of the vertices incident with f and the vertices in $\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})$. If $h > 1$ and $H_1 = C_1 \dots C_k$ is a cycle chain in $G-y$, then let A_1 consist of those vertices of $H_1 - \{a_1, b_1\}$ which are incident with f or contained in $\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})$. If $1 < s < h$ and $H_s = C_1 \dots C_k$ is a cycle chain, then let $A_s := (\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - (\{a_{s-1}, b_{s-1}\} \cup \{a_s, b_s\})$. If $1 < s = h$ and $H_s = C_1 \dots C_k$ is a cycle chain, then let $A_s := (\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - \{a_{s-1}, b_{s-1}\}$.

Note that when H_h is a cycle chain, we may choose H_h so that $x \notin A_h$. Define $\sigma(\mathcal{H}) := |\bigcup_{s=1}^{h} A_s|$. Intuitively, $\sigma(\mathcal{H})$ consists of the vertices incident with f, and those vertices which are of degree at least three in $\mathcal{H}$ (when viewed as a graph).

We wish to route our cycle through two large "parts" of $G - y$. For this purpose, we consider chains $\mathcal{I}$ and $\mathcal{J}$ in $G - y$ defined below.

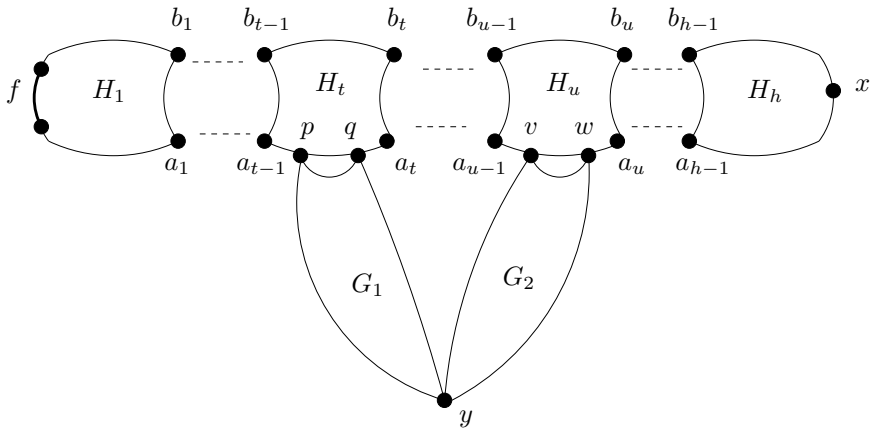


FIG. 7. *Block chains $\mathcal{H}$, $\mathcal{I}$, and $\mathcal{J}$.*

Let $\mathcal{I} := I_1 \dots I_i$ be a block chain in $G - y$ such that (i) $|V(I_1) \cap V(\mathcal{H})| = 2$, (ii) $V(\mathcal{I}) \cap V(\mathcal{H}) = V(I_1) \cap V(\mathcal{H})$, (iii) I_i is an extreme 3-block of $G - y$, and (iv) subject to (i), (ii), and (iii), $|V(\mathcal{I})|$ is maximum. We may choose the notation of e and f so that $\mathcal{I}$ is nonempty, and hence, G_1 is defined. Let $V(I_1) \cap V(\mathcal{H}) = \{p, q\}$. In this case, $\{p, q, y\}$ is a 3-cut of G, and we let G_1 denote the subgraph of G by deleting those components of $G - \{p, q, y\}$ that contain an element of $V(\mathcal{H})$. (Note that G_1 can be defined in a more direct way; however, defining it from $\mathcal{I}$ is more natural for our algorithm because we have all 3-blocks.) See Figure 7.

Since all degree two vertices in $\mathcal{H}$ are contained in some other 3-blocks of $G-y$ or are neighbors of y, $G-y$ can be covered by at most $d-1$ block chains starting from a 3-block containing f and ending with an extreme 3-block (or a cycle chain containing an extreme 3-block). Hence, we have the following.

Observation 1. $|G_1| \geq \frac{n-\sigma(\mathcal{H})}{d-1}$.

Let $\mathcal{J} := J_1 \dots J_j$ be a block chain in $G - y$ such that (i) $|V(J_1) \cap (V(\mathcal{H}) \cup V(G_1))| = 2$, (ii) $V(\mathcal{J}) \cap (V(\mathcal{H}) \cup V(G_1)) = V(J_1) \cap (V(\mathcal{H}) \cup V(G_1))$, (iii) J_j is an extreme 3-block of $G - y$, and (iv) subject to (i), (ii), and (iii), $|V(\mathcal{J})|$ is maximum. When $\mathcal{J}$ is nonempty, let $V(\mathcal{J}) \cap (V(\mathcal{H}) \cup V(G_1)) = \{v, w\}$. By the choice of G_1, $\{v, w\} \neq \{p, q\}$ and $\{v, w\} \subseteq V(\mathcal{H})$. In this case, $\{v, w, y\}$ is a 3-cut of G, and we let G_2 denote the subgraph of G by deleting those components of $G - \{v, w, y\}$ that contain an element of $V(G_1) \cup V(\mathcal{H})$. Note that $V(G_1) \cap V(G_2) \subseteq \{p, q, y\} \cap \{v, w, y\}$ and $|V(G_1) \cap V(G_2)| \leq 2$ (because $\{v, w\} \neq \{p, q\}$). See Figure 7.

By the same reasoning as for Observation 1, we have the following two observations.

Observation 2. If G_2 is defined, then $|G_2| \geq \frac{n-\sigma(\mathcal{H})-(|G_1|-1)}{d-2}$.

Observation 3. If $\sigma(\mathcal{H}) \geq |G_2|$, then $\sigma(\mathcal{H}) \geq \frac{n-|G_1|+1}{d-1}$.

Next we distinguish two cases by comparing $\sigma(\mathcal{H})$ and $|G_2|$.

Case 1. $\sigma(\mathcal{H}) \geq |G_2|$. In this case, it suffices to consider $\mathcal{H}$ and G_1. Clearly, there is some $1 \leq t \leq h$ such that $\{p, q\} \subseteq V(H_t)$, and $\{p, q\} \neq \{a_{t-1}, b_{t-1}\}$ when $t \neq 1$. Let a_0, b_0 be the vertices incident with f. We will find paths in H_s, $1 \leq s \leq h$, and a path in G_1 to form the desired cycle.

(1) If $s = 1 < t$, then there is a path P_1 from a_1 to b_1 in H_1 such that $f \in E(P_1)$ and $|E(P_1)| \geq (\frac{|A_s|}{2(d-1)})^r + 1$. If $1 < s < t$, then there exists $P_s \subseteq H_s$, consisting of

disjoint paths from $\{a_{s-1}, b_{s-1}\}$ to $\{a_s, b_s\}$, such that $|E(P_s)| \geq (\frac{|A_s|}{2(d-1)})^r + 1$.

Suppose $H_s = K_4$ or H_s is a cycle chain. By Proposition 2.4, let C_s denote a Hamilton cycle through $a_{s-1}b_{s-1}$ and a_sb_s in H_s. Since $|H_s| \geq 3$ and by Lemma 3.2,

$$|C_s| = |H_s| \geq \left(\frac{|H_s|}{2(d-1)}\right)^r + 2 \geq \left(\frac{|A_s|}{2(d-1)}\right)^r + 2.$$

If $s = 1$, then $P_1 := C_1 - a_1b_1$ gives the desired path for (1). Now assume $1 < s < t$. If $|H_s| \geq 4$, then $|C_s| = |H_s| \geq (\frac{|H_s|}{2(d-1)})^r + 3$, and if $|H_s| = 3$, then $|A_s| = 0$, and hence $|C_s| = |H_s| = (\frac{|A_s|}{2(d-1)})^r + 3$. Hence $P_s := C_s - \{a_{s-1}b_{s-1}, a_sb_s\}$ gives the desired path for (1).

Now suppose H_s is 3-connected and $H_s \neq K_4$. Then $5 \leq |H_s| < n$, and hence Theorem 2.1 holds for H_s. By Theorem 2.1(b), there is a cycle C_s through $a_{s-1}b_{s-1}$ and a_sb_s in H_s such that $|C_s| = (\frac{|H_s|}{2(d-1)})^r + 3 \geq (\frac{|A_s|}{2(d-1)})^r + 3$.

When $s \neq 1$, then $P_s := C_s - \{a_{s-1}b_{s-1}, a_sb_s\}$ is as desired, and when $s = 1$, then $P_s := C_s - a_sb_s$ gives the desired path for (1).

(2) Next we find $P_t \subseteq H_t$, and to do so, we consider three subcases.

(2a) First, assume that $1 = t = h$. We will find a path P_t from x to $\{p, q\}$ such that $f \in E(P_t)$, $pq \notin E(P_t)$ unless $pq = f$, and $|E(P_t)| \geq (\frac{\sigma(\mathcal{H})}{2(d-1)})^r + 1$.

If H_t is a cycle chain, then by Proposition 2.8, let P_t denote a path from x to $\{p, q\}$ in H_t such that $f \in E(P_t)$, $pq \notin E(P_t)$ unless $pq = f$, and $A_t \subseteq V(P_t)$. When H_t consists of only one 3-block of $G - y$, then $|E(P_t)| \geq 2 = |A_t| > (\frac{|A_t|}{2(d-1)})^r + 1 = (\frac{\sigma(\mathcal{H})}{2(d-1)})^r + 1$ (by Lemma 3.2). When H_t has at least two 3-blocks of $G - y$, then $|A_t| \geq 3$ and $|E(P_t)| \geq |A_t| - 1 \geq (\frac{|A_t|}{2(d-1)})^r + 1 = (\frac{\sigma(\mathcal{H})}{2(d-1)})^r + 1$ (by Lemma 3.2). So P_t gives the desired path for (2a).

If $H_t = K_4$, then $\sigma(\mathcal{H}) = 4$. Let P_t denote a Hamilton path from x to $\{p, q\}$ in H_t such that $f \in E(P_t)$, and $pq \notin E(P_t)$ unless $pq = f$. Then $|E(P_t)| = 3 \geq (\frac{\sigma(\mathcal{H})}{2(d-1)})^r + 2$. Hence, P_t gives the desired path for (2a).

Now assume that H_t is not a cycle chain and $H_t \neq K_4$.

If $x \in \{p, q\}$, then $f \neq pq$ since x is not incident with f. Since $5 \leq |H_t| < n$, Theorem 2.1 holds for H_t. By Theorem 2.1(b), there exists a cycle C_t in H_t such that $pq, f \in E(P_t)$ and $|C_t| \geq (\frac{|H_t|}{2(d-1)})^r + 3 = (\frac{\sigma(\mathcal{H})}{2(d-1)})^r + 3$. Hence $P_t := C_t - pq$ gives the desired path.

Assume $x \notin \{p, q\}$.

Suppose $f \neq pq$. Let H_t' be obtained from H_t by a T-transform at $\{x, pq\}$, and let x' denote the new vertex. By Lemma 3.3 and since x has degree at most $d - 1$ in H_t, H_t' is a 3-connected graph with maximum degree at most d. Since $G - y$ is not 3-connected, $|H_t| < n - 1$. Hence $5 \leq |H_t'| < n$, and Theorem 2.1 holds for H_t'. By Theorem 2.1(b), there exists a cycle C_t in H_t' such that $f, xx' \in E(C_t)$ and $|C_t| \geq (\frac{|H_t|}{2(d-1)})^r + 3 = (\frac{\sigma(\mathcal{H})}{2(d-1)})^r + 3$. Hence $P_t := C_t - x'$ gives the desired path for (2a).

Finally, assume that $f = pq$. Let $H_t' := H_t + \{px, qx\}$. Then H_t' is a 3-connected graph with maximum degree at most $d + 1$, and all vertices of degree $d + 1$ must be incident with px or pq. By Theorem 2.1(a), we can find a cycle C_t in $H_t' - p$ through xq such that $|C_t| \geq (\frac{|H_t|}{2t})^r + 2 = (\frac{\sigma(\mathcal{H})}{2t})^r + 2$, where $t \leq d - 1$ is the number of neighbors of p distinct from x and q. Hence $P_t := (C_t - qx) + \{p, pq\}$ gives the desired path for (2a).

(2b) Now assume that $1 \le t < h$. If $t = 1$, we will find a path P_t from $\{a_t, b_t\}$ to $\{p, q\}$ in H_t such that $f \in E(P_t)$, $pq \notin E(P_t)$ unless $pq = f$, and $|E(P_t)| \ge (\frac{|A_t|}{2(d-1)})^r$. If $t \ne 1$, we will find $P_t \subseteq H_t$, consisting of disjoint paths from $\{p, q\}$ and $\{a_t, b_t\}$ to $\{a_{t-1}, b_{t-1}\}$ such that $|E(P_t)| \ge (\frac{|A_t|+2}{2(d-1)})^r - 1$.

Suppose H_t is a cycle chain. If $\{a_t, b_t\} = \{p, q\}$, then by Proposition 2.4 let C_t denote a Hamilton path in H_t from a_t to b_t through $a_{t-1}b_{t-1}$. If $\{a_t, b_t\} \ne \{p, q\}$, then by Proposition 2.7, let C_t denote a path in $H_t - a_tb_t$ from $\{a_t, b_t\}$ to $\{p, q\}$ such that $a_{t-1}b_{t-1} \in E(C_t)$, $pq \notin E(C_t)$, and $A_t \subseteq V(C_t)$. From the definition of A_t and since $t < h$, $a_t \notin A_t$ and $b_t \notin A_t$. Also note that if $t \ne 1$, then $a_{t-1} \notin A_t$ or $b_{t-1} \notin A_t$. So if $t = 1$, then $|E(C_t)| \ge |A_t|$, and if $t \ne 1$, then $|E(C_t)| \ge |A_t| + 1$. Let $P_t := C_t$ if $t = 1$, and let $P_t := C_t - a_{t-1}b_{t-1}$ if $t \ne 1$. Then P_t is as desired for (2b).

If $H_t = K_4$, then let C_t denote a Hamilton path in $H_t - \{pq, a_tb_t\}$ from $\{a_t, b_t\}$ to $\{p, q\}$ through $a_{t-1}b_{t-1}$. Then $|E(C_t)| = 3 > (\frac{|A_t|}{2(d-1)})^r + 1$, and so $P_t := C_t - a_{t-1}b_{t-1}$ is as desired for (2b).

Now assume that H_t is not a cycle chain and $H_t \ne K_4$.

Suppose $\{a_t, b_t\} = \{p, q\}$. Since $5 \le |H_t| < n$, Theorem 2.1 holds for H_t. By Theorem 2.1(b), there is a cycle C_t through $a_{t-1}b_{t-1}$ and a_tb_t in H_t such that $|C_t| \ge (\frac{|H_t|}{2(d-1)})^r + 3 = (\frac{|A_t|}{2(d-1)})^r + 3$. If $t = 1$ then $P_t := C_t - a_tb_t$ gives the desired path, and if $t \ne 1$ then $P_t := C_t - \{a_{t-1}b_{t-1}, a_tb_t\}$ is as desired for (2b).

Now assume that $\{a_t, b_t\} \ne \{p, q\}$. Let H'_t be obtained from H_t by an H-transform at $\{a_tb_t, pq\}$, and let a', b' denote the new vertices. By Lemma 3.3, H'_t is a 3-connected graph with maximum degree at most d. Since $G - y$ is not 3-connected and since $\mathcal{I}$ is nonempty (when $\{p, q\}$ is defined), we have $|H_t| \le n - 3$. Hence $5 \le |H'_t| < n$. Hence Theorem 2.1 holds for H'_t. By Theorem 2.1(b), there is a cycle C_t through $a'b'$ and $a_{t-1}b_{t-1}$ in H'_t such that $|C_t| \ge (\frac{|H'_t|}{2(d-1)})^r + 3 = (\frac{|A_t|+2}{2(d-1)})^r + 3$. If $t = 1$, then $P_t := C_t - \{a, a'\}$ gives the desired path, and if $t \ne 1$ then $P_t := C_t - \{a, a', a_{t-1}b_{t-1}\}$ is as desired for (2b).

(2c) Finally, assume $1 < t = h$. We will find $P_t \subseteq H_t$, consisting of disjoint paths from x and $\{p, q\}$ to $\{a_{t-1}, b_{t-1}\}$, such that $|E(P_t)| \ge (\frac{|A_t|}{2(d-1)})^r$.

If H_t is a cycle chain, then by Proposition 2.8, let C_t denote a path in H_t from x to $\{p, q\}$ such that $a_{t-1}b_{t-1} \in E(C_t)$, $pq \notin E(P_t)$, and $A_t \subseteq V(P_t)$. Since $x, a_{t-1}, b_{t-1} \notin A_t$, $|E(C_t)| \ge |A_t| + 1 > (\frac{|A_t|}{2(d-1)})^r + 1$. Hence $P_t := C_t - a_{t-1}b_{t-1}$ is as desired for (2c).

If $H_t = K_4$, then let C_t denote a Hamilton path in $H_t - pq$ from x to $\{p, q\}$ through $a_{t-1}b_{t-1}$. Then $|E(C_t)| = 3 \ge (\frac{|H_t|}{2(d-1)})^r + 2 = (\frac{|A_t|}{2(d-1)})^r + 2$. Hence $P_t := C_t - a_{t-1}b_{t-1}$ is as desired for (2c).

Now assume that H_t is not a cycle chain and $H_t \ne K_4$.

Suppose $x \in \{p, q\}$. Since $5 \le |H_t| < n$, Theorem 2.1 holds for H_t. By Theorem 2.1(b), there is a cycle C_t through $a_{t-1}b_{t-1}$ and pq in H_t such that $|C_t| \ge (\frac{|H_t|}{2(d-1)})^r + 3 = (\frac{|A_t|}{2(d-1)})^r + 3$. Then $P_t := C_t - \{pq, a_{t-1}b_{t-1}\}$ is as desired.

Now assume that $x \notin \{p, q\}$. Recall that $\{p, q\} \ne \{a_{t-1}, b_{t-1}\}$. Let H'_t be obtained from H_t by a T-transform at $\{x, pq\}$ and let c' denote the new vertex. By Lemma 3.3, H'_t is a 3-connected graph with maximum degree at most d (because the degree of x in H_t is at most $d - 1$). Since $G - y$ is not 3-connected, $|H_t| \le n - 2$, and so, $5 \le |H'_t| < n$. Hence Theorem 2.1 holds for H'_t. By Theorem 2.1(b), there is a cycle C_t through xc' and $a_{t-1}b_{t-1}$ in H'_t such that $|C_t| \ge (\frac{|H'_t|}{2(d-1)})^r + 3 > (\frac{|A_t|}{2(d-1)})^r + 3$.

Hence $P_t := C_t - \{c', a_{t-1}b_{t-1}\}$ is as desired for (2c).

(3) For each $t+1 \le s \le h$, we will find a path $P_s \subseteq H_s$ such that $|E(P_s)| \ge (\frac{|A_s|}{2(d-1)})^r$ when $s \ne h$, $|E(P_s)| \ge (\frac{|A_s|}{2(d-1)})^r + 1$ when $s = h$, and $\bigcup_{s=t+1}^{h} P_s$ is a path from x to the end of P_t contained in $\{a_t, b_t\}$ and is otherwise disjoint from P_t.

We find P_s in the order $s = t+1, \dots, h$.

Suppose P_{s-1} is found, and the notation of $\{a_{s-1}, b_{s-1}\}$ is chosen so that a_{s-1} is an end of P_{s-1}, and assume that the notation of $\{a_s, b_s\}$ is chosen so that $a_s \notin \{a_{s-1}, b_{s-1}\}$.

First, assume that H_s is a cycle chain. If $s \ne h$, then by Proposition 2.5, let P_s denote a path in $H_s - \{a_{s-1}b_{s-1}, a_sb_s\}$ from a_{s-1} to $\{a_s, b_s\}$ such that $A_s \subseteq V(P_s)$. Since $a_{s-1}, b_{s-1}, a_s, b_s \notin A_s$, we have that $|E(P_s)| \ge |A_s|+1 \ge (\frac{|A_s|}{2(d-1)})^r+1$. If $s = h$, then by Proposition 2.6 let P_s be a path from x to a_{s-1} in $H_s - a_{s-1}b_{s-1}$ such that $A_s \subseteq V(P_s)$. Since $x, a_{s-1}, b_{s-1} \notin A_s$, we have that $|E(P_s)| \ge |A_s|+1 > (\frac{|A_s|}{2(d-1)})^r+1$.

Now assume $H_s = K_4$. If $s \ne h$, then let P_s denote a path in $H_s - \{a_{s-1}b_{s-1}, a_sb_s\}$ from a_{s-1} to a_s with $|E(P_s)| \ge 1 \ge (\frac{|A_s|}{2(d-1)})^r$. If $s = h$, then let P_s be a path in $H_s - a_{s-1}b_{s-1}$ from a_{s-1} to x with $|E(P_s)| \ge 2 > (\frac{|A_s|}{2(d-1)})^r + 1$.

Therefore, we may assume that H_s is not a cycle chain and $H_s \ne K_4$.

Suppose $s \ne h$. If $b_{s-1} = b_s$, then let $H'_s := H_s + a_{s-1}a_s$. Clearly, H'_s is 3-connected with maximum degree at most $d+1$, and the vertices of degree $d+1$ must be incident with $a_{s-1}b_{s-1}$ or a_sb_{s-1}. Thus by Theorem 2.1(a), there is a cycle C_s in $H'_s - b_{s-1}$ such that $a_{s-1}a_s \in E(C_s)$ and $|C_s| \ge (\frac{|H'_s|}{2(d-1)})^r + 2$. Let $P_s := C_s - a_{s-1}a_s$. Then $|E(P_s)| \ge (\frac{|A_s|}{2(d-1)})^r+1$. So assume $b_{s-1} \ne b_s$. Let H''_s be obtained from H_s by a T-transform at $\{b_{s-1}, a_sb_s\}$, and let a' denote the new vertex. Let $H'_s := H''_s + a_{s-1}a'$. Since $G - y$ is not 3-connected, $|H''_s| \le n-2$, and so $5 \le |H'_s| < n$. By Lemma 3.3, H''_s is a 3-connected graph with maximum degree at most $d+1$, and the vertices of degree $d+1$ must be incident with $b_{s-1}a'$ or $b_{s-1}a_{s-1}$. Thus $H'_s, a_{s-1}a', b_{s-1}$ satisfy the conditions of Theorem 2.1(a). By Theorem 2.1(a), there is a cycle C_s through $a_{s-1}a'$ in $H'_s - b_{s-1}$ such that $|C_s| \ge (\frac{|H'_s|}{2(d-1)})^r + 2$. Let $P_s := C_s - a'$. Then $|E(P_s)| \ge (\frac{|A_s|}{2(d-1)})^r$.

Now assume $s = h$. Let $H'_s := H_s + \{xb_{s-1}, xa_{s-1}\}$. Then H'_s is a 3-connected graph, the vertices x, a_{s-1}, b_{s-1} have degree at most $d+1$, and all other vertices of H'_s have degree at most d. Thus $H'_s, a_{s-1}x, b_{s-1}$ satisfy the conditions of Theorem 2.1(a). By Theorem 2.1(a), there is a cycle C'_s through $a_{s-1}x$ in $H'_s - b_{s-1}$ such that $|C'_s| \ge (\frac{|H'_s|}{2(d-1)})^r + 2$. Let $P_s := C'_s - a_{s-1}x$. Then P_s is a path from a_{s-1} to x and $|E(P_s)| \ge (\frac{|H'_s|}{2(d-1)})^r + 1 \ge (\frac{|A_s|}{2(d-1)})^r + 1$.

It is easy to see that $\bigcup_{s=t+1}^{h} P_s$ is a path from x to the end of P_t in $\{a_t, b_t\}$ and is otherwise disjoint from P_t.

(4) Let $P := \bigcup_{s=1}^{h} P_s$. We claim that P is a path from x to $\{p, q\}$, $f \in E(P)$, $pq \notin E(P)$ unless $pq = f$, and $|E(P)| \ge (\frac{\sigma(\mathcal{H})}{2(d-1)})^r + 1$.

This is obvious if $h = 1$ (by (2a)). So assume that $h \ge 2$.

Suppose $t \ne 1$. Then $|E(P_s)| \ge (\frac{|A_s|}{2(d-1)})^r + 1$ for $1 \le s \le t-1$ (by (1)), $|E(P_t)| \ge (\frac{|A_t|+2}{2(d-1)})^r - 1$ when $t \ne h$ (by (2b)), $|E(P_t)| \ge (\frac{|A_t|}{2(d-1)})^r$ when $t = h$ (by (2c)), $|E(P_s)| \ge (\frac{|A_s|}{2(d-1)})^r$ when $t+1 \le s < h$ (by (3)), and $|E(P_h)| \ge (\frac{|A_h|}{2(d-1)})^r) + 1$ when $t < h$ (by (3)). Hence we have the following:

$$\begin{aligned}|E(P)| &= \sum_{s=1}^{h} |E(P_s)| \\ &\geq \left(\sum_{s=1}^{h} \left(\frac{|A_s|}{2(d-1)}\right)^r\right) + 1 \\ &\geq \left(\frac{\sum_{s=1}^{h} |A_s|}{2(d-1)}\right)^r + 1 \quad \text{(by Lemma 3.1)} \\ &\geq \left(\frac{\sigma(\mathcal{H})}{2(d-1)}\right)^r + 1.\end{aligned}$$

Now suppose $t = 1$. Then $|E(P_t)| \geq (\frac{|A_t|}{2(d-1)})^r$ (by (2b)), $|E(P_s)| \geq (\frac{|A_s|}{2(d-1)})^r$ for $2 \leq s \leq h-1$ (by (3)), and $|E(P_h)| \geq (\frac{|A_h|}{2(d-1)})^r + 1$ (by (3)). Hence by the same argument as in the above paragraph, we have $|E(P)| \geq (\frac{\sigma(\mathcal{H})}{2(d-1)})^r + 1$. Thus, we have (4).

By (4), we may assume that the notation of $\{p, q\}$ is chosen so that P is from x to p.

(5) We claim that there is a path Q in $G_1 - q$ from p to y such that $|E(Q)| \geq (\frac{|G_1|}{2(d-1)})^r + 1$.

Note that $G_1' := G_1 + \{yp, yq, pq\}$ is a 3-connected graph.

If $G_1' = K_4$, then we can find a path Q in $G_1' - q$ from p to y such that $|E(Q)| = 2 \geq (\frac{|G_1|}{2(d-1)})^r + 1$.

Now assume that $G_1' \neq K_4$. Then Theorem 2.1 holds for G_1'. Note that all vertices of G_1' have degree at most d, except possibly y, p, q, which have degree at most $d+1$. By Theorem 2.1(a), there is a cycle C_1 through py in $G_1' - q$ such that $|C_1| \geq (\frac{|G_1|}{2(d-1)})^r + 2$. Let $Q := C_1 - py$. Then Q gives the desired path.

(6) Finally, let $C := (P \cup Q) + xy$. Then C is a cycle through e and f in G and, by (4) and (5), $|C| \geq ((\frac{\sigma(\mathcal{H})}{2(d-1)})^r + 1) + ((\frac{|G_1|}{2(d-1)})^r + 1) + 1 = (\frac{\sigma(\mathcal{H})}{2(d-1)})^r + (\frac{|G_1|}{2(d-1)})^r + 3$. Recall that $\sigma(\mathcal{H}) \geq |G_2|$ and $|G_1| \geq |G_2|$.

If $\sigma(\mathcal{H}) < |G_1|$, then

$$\begin{aligned}|C| &\geq \left(\frac{\sigma(\mathcal{H})}{2(d-1)}\right)^r + \left(\frac{|G_1|}{2(d-1)}\right)^r + 3 \\ &\geq \left((d-1)\sigma(\mathcal{H}) + \frac{|G_1|}{2(d-1)}\right)^r + 3 \quad \text{(by Lemma 3.1 and since } \sigma(\mathcal{H}) < |G_1|) \\ &\geq \left(\frac{n}{2(d-1)}\right)^r + 3 \quad \text{(by Observation 3).}\end{aligned}$$

Otherwise, $\sigma(\mathcal{H}) \geq |G_1|$. Hence,

$$\begin{aligned}|C| &\geq \left(\frac{\sigma(\mathcal{H})}{2(d-1)}\right)^r + \left(\frac{|G_1|}{2(d-1)}\right)^r + 3 \\ &\geq \left(\frac{\sigma(\mathcal{H})}{2(d-1)} + (d-1)|G_1|\right)^r + 3 \quad \text{(by Lemma 3.1 and since } \sigma(\mathcal{H}) \geq |G_1|) \\ &> \left(\frac{n}{2(d-1)}\right)^r + 3 \quad \text{(by Observation 1).}\end{aligned}$$

Case 2. $\sigma(\mathcal{H}) \le |G_2|$. In this case G_2 is defined. We will use G_1 and G_2 to find the desired cycle. Let $V(G_2) \cap V(\mathcal{H}) = \{v, w\}$. Then there exists some $1 \le u \le h$ such that $\{v, w\} \subseteq V(H_u)$, and $\{v, w\} \ne \{a_{u-1}, b_{u-1}\}$ when $u \ne 1$. Also there exists some $1 \le t \le h$ such that $\{p, q\} \subseteq V(H_t)$, and $\{p, q\} \ne \{a_{t-1}, b_{t-1}\}$ when $t \ne 1$.

(1) We claim that we can find, in $O(|V(\mathcal{H})|)$ time, a path P from x to some $z \in \{p, q\} \cup \{v, w\}$ in $\bigcup_{s=1}^{h} H_s$ for which

(i) $f \in E(P)$,

(ii) $pq \in E(P)$ or $vw \in E(P)$, and

(iii) if $pq \in E(P)$, then $z \in \{v, w\}$, and $vw \notin E(P)$ unless $vw = f$, and if $vw \in E(P)$, then $z \in \{p, q\}$, and $pq \notin E(P)$ unless $pq = f$.

To prove (1), let us assume that $t \le u$; the case $t \ge u$ can be taken care of in exactly the same way.

When $t \ne 1$, we use Lemma 3.5 to find a cycle Q' in $\bigcup_{s=1}^{t-1} H_s$ through $a_{t-1}b_{t-1}$ and f. Let $Q := Q' - a_{t-1}b_{t-1}$, which is a path from a_{t-1} to b_{t-1} through f. Let $Q = \emptyset$ when $t = 1$. We distinguish two cases.

Subcase (1a). $t < u$. By choosing the notation of $\{a_t, b_t\}$, we may assume that $(\bigcup_{s=t+1}^{h} H_s) - b_t$ contains a path X from a_t to x through vw.

If $b_t \in \{p, q\}$, then we use Lemma 3.5 to find a cycle C_t through $a_{t-1}b_{t-1}$ and a_tb_t in H_t. If $pq \notin E(C_t)$ or $b_t \in \{a_{t-1}, b_{t-1}\}$, let $P_t := C_t - \{a_{t-1}b_{t-1}, a_tb_t\}$ when $t \ne 1$, and let $P_t := C_t - a_tb_t$ when $t = 1$. Then $P := Q \cup X \cup P_t$ gives the desired path for (1) (with $z = b_t$). So assume $pq \in E(C_t)$ and $b_t \notin \{a_{t-1}, b_{t-1}\}$. Then let $P_t := C_t - \{a_{t-1}b_{t-1}, b_t\}$ when $t \ne 1$, and let $P_t := C_t - b_t$ when $t = 1$. Then $P := Q \cup X \cup P_t$ gives the desired path for (1) (with $z \in \{p, q\} - \{b_t\}$).

We may therefore assume that $b_t \notin \{p, q\}$.

Suppose H_t is not a cycle chain. Then H_t is 3-connected. Let H'_t be obtained from H_t by subdividing pq with a vertex a' and adding an edge $a'a_t$ (if not already present). Then $H'_t - b_t$ is 2-connected. By Lemma 3.5, we find a cycle C'_t through $a_{t-1}b_{t-1}$ and $a'a_t$ in $H'_t - b_t$. If $t = 1$, then let $P_t := C'_t - a'$, and if $t \ne 1$, then let $P_t := C'_t - \{a', a_{t-1}b_{t-1}\}$. Then $P := Q \cup P_t \cup X$ gives the desired path for (1).

Now assume that H_t is a cycle chain. By the structure of a cycle chain, we can, in $O(|H_t|)$ time, either find a path C_t in $H_t - a_tb_t$ from a_t to $\{p, q\}$ through $a_{t-1}b_{t-1}$, or find a path C'_t in H_t from a_t to b_t through $a_{t-1}b_{t-1}$ and pq.

If we find C_t, then let $P_t := C_t - a_{t-1}b_{t-1}$ when $t \ne 1$ and $P_t := C_t$ when $t = 1$. In this case, $P := Q \cup P_t \cup X$ gives the desired path for (1).

Assume that we find C'_t. In this case, we cannot use X. Let $P_t := C'_t$ if $t = 1$, and otherwise let $P_t := C'_t - a_{t-1}b_{t-1}$. Let $H := \bigcup_{s=t+1}^{h} H_s$. If $x \in \{v, w\}$, then find a cycle C' in H through a_tb_t and vw, and so $P := Q \cup P_t \cup (C' - \{a_tb_t, vw\})$ gives the desired path for (1). So assume that $x \notin \{v, w\}$. Let H' be obtained from H by a T-transform at $\{x, vw\}$, and let x' denote the new vertex. Then H' is a 2-connected graph. By Lemma 3.5, we find a cycle C' through a_tb_t and xx'. Now $P := Q \cup P_t \cup (C' - \{x', a_tb_t\})$ gives the desired path for (1).

Subcase (1b). $t = u$. Recall that $\{p, q\} \ne \{v, w\}$.

First, assume that $t \ne h$. We claim that there is a path Q_t in H_t from $\{a_t, b_t\}$ to some $z \in \{p, q\} \cup \{v, w\}$ such that (i) $a_{t-1}b_{t-1} \in E(Q_t)$, (ii) $pq \in E(Q_t)$ or $vw \in E(Q_t)$, and (iii) if $pq \in E(Q_t)$, then $z \in \{v, w\}$, and $vw \notin E(Q_t)$ unless $vw = a_{t-1}b_{t-1}$, and if $vw \in E(Q_t)$, then $z \in \{p, q\}$, and $pq \notin E(Q_t)$ unless $pq = a_{t-1}b_{t-1}$. This is easy to see if H_t is a cycle chain, and otherwise, it follows from Lemma 3.6. Assume without loss of generality that $a_t \in V(Q_t)$. In $(\bigcup_{s=t+1}^{h} H_s) - b_t$, we find a path R from a_t to x. Now $P := Q \cup Q_t \cup R$ gives the desired path for (1).

Now assume that $t = h$. We note that there is a path Q_t in H_t from x to $z \in \{p, q\} \cup \{v, w\}$ such that (i) $a_{t-1}b_{t-1} \in E(Q_t)$, (ii) $pq \in E(Q_t)$ or $vw \in E(Q_t)$, and (iii) if $pq \in E(Q_t)$, then $z \in \{v, w\}$, and $vw \notin E(Q_t)$ unless $vw = a_{t-1}b_{t-1}$, and if $vw \in E(Q_t)$, then $z \in \{p, q\}$, and $pq \notin E(Q_t)$ unless $pq = a_{t-1}b_{t-1}$. This is easy to see if H_t is a cycle chain, and otherwise, it follows from Lemma 3.7. Now $P := Q \cup Q_t$ gives the desired path for (1).

Assume that $vw \in E(P)$ in (1) (the case $pq \in E(P)$ is similar), and assume p is an end of P.

(2) Note that $G_1' := G_1 + \{yp, yq, pq\}$ is a 3-connected graph, vertices y, p, q have degree at most $d + 1$ in G_1', and all other vertices of G_1' have degree at most d. If $G_1' = K_4$, then we can find a path P_1 from p to y in $G_1' - q$ such that $|E(P_1)| = 2 \geq (\frac{|G_1|}{2(d-1)})^r + 1$. If $G_1' \neq K_4$, then Theorem 2.1 holds for G_1'. By Theorem 2.1(a), there is a cycle C_1 through py in $G_1' - q$ such that $|C_1| \geq (\frac{|G_1|}{2t_1})^r + 2$, where t_1 is the number of neighbors of q in G_1' distinct from p and y. Let $P_1 := C_1 - py$. Then P_1 is a path from p to y in $G_1' - q$. Since $t_1 \leq d - 1$, we have $|E(P_1)| \geq (\frac{|G_1|}{2(d-1)})^r + 1$.

(3) Note that $G_2' := G_2 + \{yv, yw, vw\}$ is a 3-connected graph, vertices y, v, w have degree at most $d + 1$ in G_2', and all other vertices of G_2' have degree at most d. If $G_2' = K_4$, then we can find a path P_2 from v to w in $G_2' - y$ such that $|E(P_2)| = 2 \geq (\frac{|G_2|}{2(d-1)})^r + 1$. If $G_2' \neq K_4$, then Theorem 2.1 holds for G_2'. By Theorem 2.1(a), there is a cycle C_2 through vw in $G_2' - y$ such that $|C_2| \geq (\frac{|G_2'|}{2t_2})^r + 2$, where t_2 is the number of neighbors of y in G_2' distinct from v and w. Let $P_2 := C_2 - vw$. Then P_2 is a path from v to w in $G_2' - y$. Since $t_2 \leq d - 1$, we have $|E(P_2)| \geq (\frac{|G_2|}{2(d-1)})^r + 1$.

Let $C := ((P - vw) \cup P_1 \cup P_2) + e$. Then C is a cycle through e and f in G and

$$\begin{aligned}
|C| &= |E(P)| + |E(P_1)| + |E(P_2)| + 1 \\
&\geq \left(\frac{|G_1|}{2(d-1)}\right)^r + \left(\frac{|G_2|}{2(d-1)}\right)^r + 3 \quad \text{(by (2) and (3))} \\
&\geq \left(\frac{|G_1|}{2(d-1)} + (d-1)|G_2|\right)^r + 3 \quad \text{(by Lemma 3.1 and since } |G_1| \geq |G_2|\text{)} \\
&\geq \left(\frac{n}{2(d-1)}\right)^r + 3,
\end{aligned}$$

where the final inequality holds because of Observation 2 and since $|G_2| \geq \sigma(\mathcal{H})$. □

Next we show that the above proof gives an $O(|E(G)|)$ algorithm which reduces Theorem 2.1(b) to (a) and (b) of the same theorem (for smaller graphs).

ALGORITHM TWOEDGE. Let n, d, r, G, e, f be as in Lemma 5.1.

1. *Preprocessing.* Replace G with a 3-connected spanning subgraph of G with $O(|G|)$ edges. (This can be done in $O(|E(G)|)$ time using Lemma 3.4.)
2. If e is adjacent to f, then let $e = xz$ and $f = yz$. It suffices to find a cycle C' through xy in $G' := (G + xy) - z$ such that $|C'| \geq (\frac{|G'|}{2t})^r + 2$, where t is the number of neighbors of z in G' distinct from x and y. That is, we reduce Theorem 2.1(b) for G, e, f to Theorem 2.1(a) for G', xy, z. We apply Algorithm Avoidvertex to G', xy, z. (By Proposition 4.2, we can, in $O(|E(G)|)$ time, either find the desired cycle C' or reduce it to (a) or (c) for smaller graphs. Moreover, each smaller graph contains a vertex that does not belong to any other smaller graph.)
3. Now assume that e is not adjacent to f, and let $e = xy$. Decompose $G - y$ into 3-connected components. (This can be done in $O(|G|)$ time using Theorem

2.2.)

4. Suppose there is only one 3-connected component of $G-y$. Then $G-y$ is 3-connected, and let y' denote a neighbor of y distinct from x. Let $G' := (G-y)+xy'$ and $e'=xy'$. To find the desired cycle through e and f in G, it suffices to find a cycle C' through e' and f in G' such that $|C'| \geq (\frac{|G'|}{2(d-1)})^r+3$. Thus we reduce Theorem 2.1(b) for G, e, f to Theorem 2.1(b) for G', e', f, with $|G'|<|G|$. (This reduction can be done in constant time.)
5. Now assume that $G-y$ has at least two 3-connected components. Find the block chain $\mathcal{H}=H_1\dots H_h$ such that $f\in E(H_1)$, $x\in V(H_h)-V(H_{h-1})$, and f is not incident with both vertices in $V(H_1)\cap V(H_2)$. Find G_1 and G_2 as in the proof of Lemma 5.1. (This can be done in $O(|G|)$ time.)
6. Suppose $\sigma(\mathcal{H})>|G_2|$.
   - Assume $1\leq s\leq t-1$. We need to find P_s as in (1) of Case 1 in the proof of Lemma 5.1. If $H_s=K_4$ or H_s is a cycle chain then we find P_s, and otherwise, we need to find a cycle C_s through $a_{s-1}b_{s-1}$ and a_sb_s in H_s such that $|C_s|\geq(\frac{|H_s|}{2(d-1)})^r+3$. So either we find P_s in $O(|H_s|)$ time or we reduce the problem of finding P_s to Theorem 2.1(b) for $H_s, a_{s-1}b_{s-1}, a_sb_s$ in constant time.
   - We need to find $P_t\subseteq H_t$ as in (2) of Case 1 in the proof of Lemma 5.1. If H_t is a cycle chain or $H_t=K_4$ then we find $P_t\subseteq H_t$ as in (2) of Case 1 in the proof of Lemma 5.1. (This can be done in $O(|H_t|)$ time.) If H_t is not a cycle chain and $H_t\neq K_4$, then we reduce the problem of finding P_t to the following: Theorem 2.1(b) for H_t, f, pq or H'_t, f, xx'; Theorem 2.1(a) for H'_t, px, q (as in (2a) of Case 1); Theorem 2.1(b) for $H_t, a_{t-1}b_{t-1}, a_tb_t$ or $H'_t, a_{t-1}b_{t-1}, a'b'$ (as in (2b) of Case 1); Theorem 2.1(b) for $H_t, a_{t-1}b_{t-1}, pq$ or $H'_t, a_{t-1}b_{t-1}, xc'$ (as in (2c) of Case 1). (This reduction can be done in constant time.)
   - Suppose $t+1\leq s\leq h$. We need to find P_s as in (3) of Case 1 in the proof of Lemma 5.1. If $H_s=K_4$ or H_s is a cycle chain, we find a path P_s. (This can be done in $O(|H_s|)$ time.) If $H_s\neq K_4$ and H_s is not a cycle chain, then we reduce the problem of finding P_s to the following: Theorem 2.1(a) for $H'_s, a_{s-1}a_s, b_{s-1}$, or $H'_s, a_{s-1}a', b_{s-1}$, or $H'_s, a_{s-1}x, b_{s-1}$. (This reduction can be done in constant time.)
   - Let $G'_1:=G_1+\{yp, yq, pq\}$. We need to find a path Q in G'_1 as in (5) of Case 1 in the proof of Lemma 5.1. If $G'_1=K_4$, we find a path Q in $O(|G_1|)$ time, and otherwise, we reduce the problem of finding Q to Theorem 2.1(a) for G'_1, py, q, in constant time.

   (The operations in step 6 can be done in $O(|G|)$ time. Also each 3-connected graph reduced from H_s's or G_1 contains a vertex which does not belong to any other 3-connected graphs reduced from H_s's or G_1.)
7. Now assume $\sigma(\mathcal{H})\leq|G_2|$.
   - First, we find H_t and H_u such that $\{p,q\}\subseteq V(H_t)$, $\{p,q\}\neq\{a_{t-1},b_{t-1}\}$ when $t\neq 1$, $\{v,w\}\subseteq V(H_u)$, and $\{v,w\}\neq\{a_{u-1},b_{u-1}\}$ when $u\neq 1$. (This can be done in $O(|G|)$ time by searching the 3-connected components of $G-y$.)
   - Assume that $t\leq u$ ($u\geq t$ can be treated similarly). Find a path P in $\bigcup_{s=1}^{t}H_s$ from x to $\{p,q\}$ (or $\{v,w\}$) through f and vw (or pq). (This can be done in $O(|G|)$ time as in (1) of Case 2 in the proof of Lemma 5.1.)

- Assume P is from x to p and through f and vw. If $G'_1 = K_4$, then we find a path P_1 in $G'_1 - q$ from p to y of length 2. If $G'_1 \neq K_4$, then we need to apply Theorem 2.1(a) to G'_1, yp, q. (This reduction can be done in constant time, as in (2) of Case 2 in the proof of Lemma 5.1.)
- If $G'_2 = K_4$, then find a path P_2 in $G'_2 - y$ from v to w of length 2. If $G'_2 \neq K_4$, then we need to apply Theorem 2.1(a) to G'_2, vw, y. (Again, this can be done in constant time, as in (3) of Case 2 in the proof of Lemma 5.1.)

To summarize, we have the following.

PROPOSITION 5.2. *Given G, e, f, n, d, r as in Lemma 5.1, we can, in $O(|E(G)|)$ time, either*

(1) *find a cycle C through e and f in G such that $|C| \geq (\frac{|G|}{2(d-1)})^r + 3$, or*
(2) *reduce Theorem 2.1(b) for G, e, f to (a) or (b) of the same theorem for smaller 3-connected graphs.*

Moreover, any smaller graph in (2) comes from a 3-connected 3-block of $G - y$ that is not K_4. Hence, any smaller graph in (2) contains a vertex that does not belong to any other smaller graph in (2).

6. Cycles through one edge. In this section, we show how to reduce Theorem 2.1(c), in linear time, to (a), (b), or (c) of the same theorem for smaller graphs. As in the previous two sections, we state the reduction as a lemma.

LEMMA 6.1. *Let $n \geq 6$ and $d \geq 3$ be integers, let $r = \log_{2(d-1)^2+1} 2$, and assume that Theorem 2.1 holds for graphs with at most $n-1$ vertices. Let G be a 3-connected graph on n vertices, and assume that the maximum degree of G is at most d. Then for any $e \in E(G)$ there is a cycle C through e in G such that $|C| \geq n^r + 3$.*

Proof. Let $e = xy \in E(G)$, and consider $G - y$.

If $G - y$ is 3-connected, then let y' be a neighbor of y other than x. Clearly, $G' := (G - y) + xy'$ is a 3-connected graph with maximum degree at most d. Since $5 \leq |G'| < n$, Theorem 2.1 holds for G'. By Theorem 2.1(c), there is a cycle C' through xy' in G' such that $|C'| \geq (n-1)^r + 3$. Now let $C := (C' - xy') + \{y, xy, yy'\}$. Then C is a cycle through xy in G and

$$\begin{aligned} |C| &= |C'| + 1 \\ &\geq (n-1)^r + 1 + 3 \\ &\geq n^r + 3 \quad \text{(by Lemma 3.1).} \end{aligned}$$

Therefore, we may assume that $G - y$ is not 3-connected. Since G is 3-connected, $G - y$ is 2-connected. By Theorem 2.2, we can decompose $G - y$ into 3-connected components.

First, let us consider the case where all 3-blocks of $G - y$ are cycles. Let $\mathcal{I} = I_1 \dots I_i$ be a block chain in $G - y$ such that (i) $x \in V(I_1) - V(I_2)$, (ii) I_i is an extreme 3-block of $G - y$, and (iii) subject to (i) and (ii), $|V(\mathcal{I})|$ is maximum. For convenience, let $B := I_1$. Then $|V(\mathcal{I})| \geq \frac{(n-1)-|B|}{t-1} + |B| = \frac{n+(t-2)|B|-1}{t-1}$, where t is the number of extreme 3-blocks of $G - y$ distinct from I_1. So $n \geq t + 4$ (since $|B| \geq 3$) and $t \leq d - 1$. It is easy to see that there is some $y' \in V(\mathcal{I}) - \{x\}$ such that $\bigcup_{s=1}^{i} I_s$ contains a Hamilton path P from x to y' and G has a path Q from y' to y disjoint from $V(\mathcal{I}) - \{y\}$. (Note that P and y' can be found in $O(|G|)$ time.) Let $C := (P \cup Q) + \{y, xy, yy'\}$. Then $|C| \geq |V(\mathcal{I})| + 1 \geq \frac{n+(t-2)|B|-1}{t-1} + 1$. Next we show that $|C| - 3 \geq n^r$. Note that $|C| - 3 \geq \frac{n+(t-2)|B|-1}{t-1} - 2 \geq \frac{n+t-5}{t-1}$ (since $|B| \geq 3$).

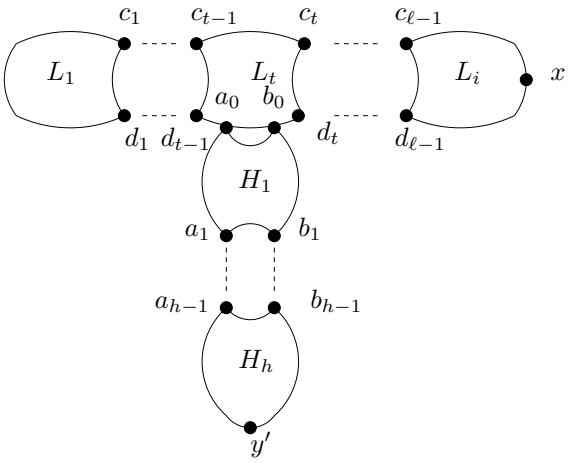


FIG. 8. *Block chains $\mathcal{L}$ and $\mathcal{H}$.*

One can prove that $\frac{x+t-5}{t-1} - x^r$ is an increasing function when $x \geq 2(d-1)^2+1$. Hence when $n \geq 2(d-1)^2+1$, $\frac{n+t-5}{t-1} \geq n^r$. Now if $t+4 \leq n \leq 2(d-1)^2$, then $\frac{n+t-5}{t-1} > 2 > n^r$. Therefore, $|C|-3 \geq n^r$, and so $|C| \geq n^r+3$.

Hence, we may assume that some 3-block of $G-y$ is 3-connected. Let L denote a 3-connected 3-block of $G-y$ with $|L|$ maximum. Then $|L| \geq 4$. Let $\mathcal{L} := L_1 \dots L_\ell$ denote a block chain in $G-y$ such that $L_1 = L$ and $x \in V(L_\ell) - V(L_{\ell-1})$, where $\ell \geq 1$. See Figure 8.

Let $V(L_i) \cap V(L_{i+1}) = \{c_i, d_i\}$ for $1 \leq i \leq \ell-1$. For each $1 \leq i \leq \ell$, we define B_i as follows: if L_i is 3-connected, then $B_i := V(L_i)$; if $i < \ell$ and $L_i = C_1 \dots C_k$ is a cycle chain, then $B_i = (\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - (\{c_{i-1}, d_{i-1}\} \cup \{c_i, d_i\})$; if $L_\ell = C_1 \dots C_k$ is a cycle chain, then B_ℓ consists of x and the vertices in $(\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - \{c_{\ell-1}, d_{\ell-1}\}$. Define $\sigma(\mathcal{L}) := |\bigcup_{s=1}^{\ell} B_s|$.

If $V(\mathcal{L}) = V(G) - \{y\}$, then let $\mathcal{H} = \emptyset$. Otherwise, let $\mathcal{H} := H_1 \dots H_h$ denote a block chain in $G-y$ such that (i) $|V(\mathcal{H}) \cap V(\mathcal{L})| = 2$, (ii) $V(\mathcal{H}) \cap V(\mathcal{L}) = V(H_1) \cap V(\mathcal{L}) \neq V(H_1) \cap V(H_2)$, (iii) H_h is an extreme 3-block of $G-y$. See Figure 8. Let $V(H_i) \cap V(H_{i+1}) = \{a_i, b_i\}$, $1 \leq i \leq h-1$. Let a_0, b_0 denote the vertices in $V(\mathcal{H}) \cap V(\mathcal{L})$. If $\mathcal{H} = \emptyset$, then let y' denote a neighbor of y distinct from x and let $a_0 = b_0 = y'$. If $\mathcal{H} \neq \emptyset$, then let y' be a neighbor of y in $V(H_h) - \{a_{h-1}, b_{h-1}\}$. For each $1 \leq s \leq h$, we define A_s as follows: if H_s is 3-connected, then $A_s := V(H_s)$; if $h = 1$ and $H_1 = C_1 \dots C_k$ is a cycle chain, then $A_1 := (\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) \cup (\{a_0, b_0\} - B_t)$; if $h > 1$ and $H_1 = C_1 \dots C_k$ is a cycle chain, then A_1 is the union of $\{a_0, b_0\} - B_t$ and $(\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - \{a_1, b_1\}$; if $1 < s < h$ and $H_s = C_1 \dots C_k$ is a cycle chain, then $A_s = (\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - \{a_{s-1}, b_{s-1}, a_s, b_s\}$; if $h > 1$ and $H_h = C_1 \dots C_k$ is a cycle chain, then A_h consists of y' and those vertices in $(\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - \{a_{h-1}, b_{h-1}\}$. Define $\sigma(\mathcal{H}) := |\bigcup_{s=1}^{h} A_s|$.

We choose $\mathcal{H} = H_1 \dots H_h$ so that, subject to (i)–(iii), $\sigma(\mathcal{H})$ is maximum. Without loss of generality, we may assume that, for some $1 \leq t \leq \ell$, $a_0, b_0 \in V(L_t)$ and $\{a_0, b_0\} \neq \{c_{t-1}, d_{t-1}\}$ (or $y' \notin \{c_{t-1}, d_{t-1}\}$ if $a_0 = b_0 = y'$) when $t \geq 2$.

Note that each vertex in $(\bigcup_{i=1}^{\ell}(V(L_i) - B_i)) \cup (\bigcup_{s=1}^{h}(V(H_s) - A_s))$ either appears in some other block chain in $G-y$ or is adjacent to y. By the choice of $\mathcal{H}$ and

since there are at most $d-1$ extreme blocks of $G-y$ not containing x, we have $\sigma(\mathcal{H}) \geq \frac{(n-1)-\sigma(\mathcal{L})}{d-1}$. Hence we have the following.

Observation. $\sigma(\mathcal{L}) + (d-1)\sigma(\mathcal{H}) \geq n-1$.

We consider three cases.

Case 1. $\ell = 1$. In this case, $\sigma(\mathcal{L}) = |L_1|$ (because L_1 is not a cycle chain). Since $G-y$ is not 3-connected, $\mathcal{H} \neq \emptyset$.

(1) First, we find a path P in $\mathcal{L} - a_0b_0$ from x to $\{a_0, b_0\}$ such that $|E(P)| \geq (\sigma(\mathcal{L})+1)^r + 1$.

If $L_1 = K_4$, then we can find a Hamilton path P from x to $\{a_0, b_0\}$ in $L_1 - a_0b_0$. Hence by Lemma 3.2, $|E(P)| = 3 \geq |L_1|^r + 2 \geq (\sigma(\mathcal{L})+1)^r + 1$. So assume that $L_1 \neq K_4$.

If $x \in \{a_0, b_0\}$, then by Theorem 2.1(c), there is a cycle C_1 through a_0b_0 in L_1 such that $|C_1| \geq |L_1|^r + 3$. Now $P := C_1 - a_0b_0$ is a path from x to $\{a_0, b_0\}$ in $L_1 - a_0b_0$ and $|E(P)| \geq (\sigma(\mathcal{L})^r + 2(\sigma(\mathcal{L})+1)^r + 1$ (by Lemma 3.1).

Assume $x \notin \{a_0, b_0\}$. Let L_1' denote the graph obtained from L_1 by a T-transform at $\{x, a_0b_0\}$, and let x' denote the new vertex. By Lemma 3.3 and because x has degree at most $d-1$ in L_1, L_1' is a 3-connected graph with maximum degree at most d. Note that $5 \leq |L_1'| < n$. Thus Theorem 2.1 holds for L_1'. By Theorem 2.1(c), there is a cycle C_1 through xx' in L_1' such that $|C_1'| \geq |L_1'|^r + 3 = (|L_1|+1)^r + 3$. Now $P := C_1 - x'$ gives the desired path.

Without loss of generality, we may assume that the path P found in (1) is from x to a_0.

(2) For $i = 1, \ldots, h-1$, we find paths Q_i from $\{a_{i-1}, b_{i-1}\}$ to $\{a_i, b_i\} - \{a_{i-1}, b_{i-1}\}$ in H_i such that

(i) $\bigcup_{i=1}^{h-1} Q_i$ is a path from a_0 to $\{a_{h-1}, b_{h-1}\}$,

(ii) if H_i is 3-connected, then $|E(Q_i)| \geq (\frac{|A_i|}{2(d-1)})^r$,

(iii) if H_i is a cycle chain, then $|E(Q_i)| \geq |A_i| + 1$.

Suppose that, for some $1 \leq i \leq h-1$, we have found paths Q_j, $1 \leq j \leq i-1$, from $\{a_{j-1}, b_{j-1}\}$ to $\{a_j, b_j\}$ in H_j such that $\bigcup_{j=1}^{i-1} Q_j$ is a path from a_0 to $\{a_{i-1}, b_{i-1}\}$ and (ii)–(iii) above are satisfied for H_j, Q_j, A_j. For ease of presentation, we may assume that a_{i-1} is an end of $\bigcup_{j=1}^{i-1} Q_j$.

If $H_i = K_4$, then we find a path Q_i in $H_i - b_{i-1}$ from a_{i-1} to $\{a_i, b_i\}$ such that $|E(Q_i)| \geq 2$. Clearly, $|E(Q_i)| \geq 2 \geq (\frac{|A_i|}{2(d-1)})^r$.

If H_i is a cycle chain, then by Proposition 2.5 let Q_i be a path in $H_i - \{a_{i-1}b_{i-1}, a_ib_i\}$ from a_{i-1} to $\{a_i, b_i\}$ such that $A_i \subseteq V(Q_i)$. Since $(\{a_{i-1}, b_{i-1}\} \cup \{a_i, b_i\}) \cap A_i = \emptyset$, we have $|E(Q_i)| \geq |A_i| + 1$.

Now assume that H_i is not a cycle chain and $H_i \neq K_4$.

If $b_{i-1} \in \{a_i, b_i\}$, then assume $b_{i-1} = b_i$ (by choosing the notation of $\{a_i, b_i\}$), and let $H_i' := H_i + a_{i-1}a_i$. Clearly, H_i' is 3-connected with maximum degree at most $d+1$, and the possible vertices of degree $d+1$ are incident with $a_{i-1}b_{i-1}$ or a_ib_{i-1}. By Theorem 2.1(a), there is a cycle D_i through $a_{i-1}a_i$ in $H_i' - b_{i-1}$ such that $|E(D_i)| \geq (\frac{|A_i|}{2t_i})^r + 2$, where t_i is the number of neighbors of b_{i-1} in H_i' distinct from a_{i-1} and a_i. So $t_i \leq d-1$. Let $Q_i := D_i - a_{i-1}a_i$. Then $|E(Q_i)| \geq (\frac{|A_i|}{2(d-1)})^r + 1$.

Hence, assume $b_{i-1} \notin \{a_i, b_i\}$. Let H_i'' be obtained from H_i by a T-transform at $\{b_{i-1}, a_ib_i\}$, and let a denote the new vertex. By Lemma 3.3, H_i'' is a 3-connected graph. Let $H_i' := H_i'' + a_{i-1}a$. Then $5 \leq |H_i'| < n$. Also a_{i-1}, b_{i-1}, a have degrees at most $d+1$ in H_i', and all other vertices have degree at most d in H_i'. Thus by Theorem 2.1(a), there is a cycle D_i through $a_{i-1}a$ in $H_i' - b_{i-1}$ such that $|D_i| \geq (\frac{|H_i'|}{2t_i})^r + 2$,

where t_i is the number of neighbors of b_{i-1} in H_i' distinct from a and a_{i-1}. Thus $t_i \le d-1$. Let $Q_i := D_i - a$. Then Q_i is a path in $H_i - b_{i-1}$ between a_{i-1} and $\{a_i, b_i\}$ and $|E(Q_i)| \ge (\frac{|H_i|}{2(d-1)})^r$.

(3) We find a path Q_h in H_h from $\{a_{h-1}, b_{h-1}\}$ to y' such that

(i) $\bigcup_{i=1}^{h} Q_i$ is a path from a_0 to y',

(ii) if H_h is 3-connected, then $|E(Q_h)| \ge (\frac{|A_h|}{2(d-1)})^r + 1$,

(iii) if H_h is a cycle chain, then $|E(Q_h)| \ge |A_h| + 1$.

For ease of presentation, let us assume that $\bigcup_{i=1}^{h-1} Q_i$ is from a_0 to a_{h-1}.

If H_h is a cycle chain, then by Proposition 2.6, let Q_h denote a path from a_{h-1} to y' in $H_h - b_{h-1}$ such that $A_h \subseteq V(Q_h)$. Since a_{h-1}, b_{h-1}, and $y' \notin A_h$, we have (iii).

If $H_h = K_4$, then let Q_h be a Hamilton path from a_{h-1} to y' in $H_h - b_{h-1}$. Then $|E(Q_h)| = 2 \ge (\frac{|H_h|}{2(d-1)})^r + 1 = (\frac{|A_h|}{2(d-1)})^r + 1$, and (ii) holds.

Now assume that H_h is not a cycle chain and $H_h \ne K_4$. Let $H_h' := H_h + \{a_{h-1}y', b_{h-1}y'\}$. Then H_h' is 3-connected, the vertices a_{h-1}, b_{h-1}, y' have degree at most $d+1$ in H_h', and all other vertices have degree at most d in H_h'. By Theorem 2.1(a), there is a cycle D_h through $a_{h-1}y'$ in $H_h' - b_{h-1}$ such that $|D_h| \ge (\frac{|H_h|}{2(d-1)})^r + 2$. Let $Q_h := D_h - a_{h-1}y'$. Then $|E(Q_h)| \ge (\frac{|A_h|}{2(d-1)})^r + 1$, and we have (ii).

(4) Let $C := (P \cup (\bigcup_{i=1}^{h} Q_i)) + \{y, xy, yy'\}$. Now C is a cycle in G through xy and, by (1)–(3), we have

$$|C| \ge (\sigma(\mathcal{L}) + 1)^r + 1 + \left(\sum |A_i|\right) + \left(\sum \left(\frac{|A_i|}{2(d-1)}\right)^r\right) + 2,$$

where the first summation is over all cycle chains H_i and the second summation is over all 3-connected H_i. Note that each $|A_i|$ in the first summation can be written as $1^r + \cdots + 1^r$ ($|A_i|$ times), and this allows us to apply Lemma 3.1 in the following inequalities. Hence

$$\begin{aligned} |C| &\ge \left(\sigma(\mathcal{L}) + 1 + (d-1)\sum_{i=1}^{h} |A_i|\right)^r + 3 \quad \text{(by Lemma 3.1)} \\ &= (\sigma(\mathcal{L}) + 1 + (d-1)\sigma(\mathcal{H}))^r + 3 \\ &\ge n^r + 3 \quad \text{(by the observation preceding Case 1).} \end{aligned}$$

Case 2. $1 = t < \ell$. Recall that if $\mathcal{H} = \emptyset$, then $a_0 = b_0 = y'$, and $\{a_0, b_0\} = \{y'\}$.

(1) We find a path P_1 from $\{a_0, b_0\}$ to $\{c_1, d_1\}$ in $L_1 - \{a_0b_0, c_1d_1\}$ such that $|E(P_1)| \ge (|B_1| + 2)^r$. (Note that $|B_1| = |L_1|$ by definition.)

If $L_1 = K_4$, then we can find a Hamilton path P_1 from $\{a_0, b_0\}$ to $\{c_1, d_1\}$ in $L_1 - \{a_0b_0, c_1d_1\}$ (or in $L_1 - c_1d_1$ when $\mathcal{H} = \emptyset$). Thus, $|E(P_1)| = 3 \ge |L_1|^r + 1 = |B_1|^r + 1 > (|B_1| + 2)^r$ (by Lemma 3.1).

Now assume that $L_1 \ne K_4$. If $\{a_0, b_0\} \subseteq \{c_1, d_1\}$, then by Theorem 2.1(c) there is a cycle C_1 through c_1d_1 in L_1 such that $|C_1| \ge |L_1|^r + 3$. Let $P_1 := C_1 - c_1d_1$; then P_1 is a path between c_1 and $d_1 \in \{a_0, b_0\}$, and $|E(P_1)| \ge |L_1|^r + 2 = |B_1|^r + 2 > (|B_1| + 2)^r$ (by Lemma 3.1).

Assume that $\{a_0, b_0\} \not\subseteq \{c_1, d_1\}$. If $\mathcal{H} = \emptyset$, then let L_1' be obtained from L_1 by a T-transform at $\{y', c_1d_1\}$, let $a = y'$, and let c denote the new vertex. If $\mathcal{H} \ne \emptyset$, then let L_1' be obtained from L_1 by an H-transform at $\{a_0b_0, c_1d_1\}$, and let a, c denote the new vertices with a adjacent to a_0 and b_0 and c adjacent to c_1 and d_1. By Lemma

3.3, L_1' is a 3-connected graph with maximum degree at most d. Since $5 \le |L_1'| < n$, Theorem 2.1 holds for L_1'. By Theorem 2.1(c), there is a cycle C_1 in L_1' through ac such that $|C_1| \ge |L_1'|^r + 3$. If $\mathcal{H} = \emptyset$, let $P_1 := C_1 - c$; then P_1 is a path from $\{a_0, b_0\}$ to $\{c_1, d_1\}$ in L_1 and $|E(P_1)| \ge |L_1'|^r + 1 = (|L_1| + 1)^r + 1 \ge (|L_1| + 2)^r = (|B_1| + 2)^r$ (by Lemma 3.1). If $\mathcal{H} \ne \emptyset$ let $P_1 := C_1 - \{a, c\}$; then P_1 is a path from $\{a_0, b_0\}$ to $\{c_1, d_1\}$ in L_1 and $|E(P_1)| \ge |L_1'|^r = (|L_1| + 2)^r = (|B_1| + 2)^r$.

Without loss of generality, assume that the notation of $\{a_0, b_0\}$ and $\{c_1, d_1\}$ is chosen so that P_1 is between c_1 and a_0.

(2) For $i = 2, \dots, \ell - 1$, we find paths P_i from $\{c_{i-1}, d_{i-1}\}$ to $\{c_i, d_i\}$ in L_i such that

(i) $\bigcup_{i=2}^{\ell-1} P_i$ is a path from c_1 to $\{c_{\ell-1}, d_{\ell-1}\}$,

(ii) if L_i is 3-connected, then $|E(P_i)| \ge (\frac{|B_i|}{2(d-1)})^r$, and

(iii) if L_i is a cycle chain, then $|E(P_i)| \ge |B_i| + 1$.

Assume that, for some $2 \le i \le \ell - 1$, we have found paths P_j, $1 \le j \le i - 1$, from $\{c_{j-1}, d_{j-1}\}$ to $\{c_j, d_j\}$ in L_j such that $\bigcup_{j=2}^{i-1} P_j$ is a path from c_1 to $\{c_{i-1}, d_{i-1}\}$ and (ii) and (iii) are satisfied for L_j, B_j, P_j. Without loss of generality, assume that c_{i-1} is the end of $\bigcup_{j=2}^{i-1} P_j$ other than c_1.

If L_i is a cycle chain, then by Proposition 2.5 let P_i be a path from c_{i-1} to $\{c_i, d_i\}$ in $L_i - \{c_{i-1}d_{i-1}, c_i d_i\}$ such that $B_i \subseteq V(P_i)$. Then, since $c_{i-1}, d_{i-1}, c_i, d_i \notin B_i$, we see that $|E(P_i)| \ge |B_i| + 1$, and we have (iii).

Now assume that L_i is not a cycle chain.

If $c_{i-1} \in \{c_i, d_i\}$, then, by choosing the notation of $\{c_i, d_i\}$, we may assume $c_i \notin \{c_{i-1}, d_{i-1}\}$. If $L_i = K_4$, then let P_i be a path from c_{i-1} to c_i in $L_i - d_{i-1}$ such that $|E(P_i)| = 2 \ge (\frac{|B_i|}{2(d-1)})^r$. If $L_i \ne K_4$, then let $L_i' := L_i + c_i d_{i-1}$. By Theorem 2.1(a), there is a cycle C_i through $c_{i-1}c_i$ in $L_i' - d_{i-1}$ such that $|C_i| \ge (\frac{|L_i'|}{2(d-1)})^r + 2 = (\frac{|B_i|}{2(d-1)})^r + 2$. Let $P_i := C_i - c_i c_{i-1}$; then P_i is a path from c_{i-1} to c_i in $L_i - d_{i-1}$, $|E(P_i)| \ge (\frac{|B_i|}{2(d-1)})^r + 1$, and we have (ii).

Assume that $c_{i-1} \notin \{c_i, d_i\}$. Let L_i'' be obtained from L_i by a T-transform at $\{c_{i-1}, c_i d_i\}$, and let c denote the new vertex. Let $L_i' := L_i'' + d_{i-1}c$. By Lemma 3.3, L_i' is 3-connected. Note that c, c_i, d_{i-1} have degree at most $d + 1$ in L_i', and all other vertices have degree at most d in L_i'. By Theorem 2.1(a), there is a cycle C_i through $c_{i-1}c$ in $L_i' - d_{i-1}$ such that $|C_i| \ge (\frac{|L_i'|}{2(d-1)})^r + 2$. Let $P_i := C_i - c_{i-1}'$. Then P_i is a path from c_{i-1} to $\{c_i, d_i\}$ in $L_i - d_{i-1}$ and $|E(P_i)| \ge (\frac{|B_i|}{2(d-1)})^r$, and we have (iii).

(3) We find a path P_ℓ from $\{c_{\ell-1}, d_{\ell-1}\}$ to x in L_ℓ such that

(i) $\bigcup_{i=2}^{\ell} P_i$ is a path from c_1 to x,

(ii) if L_ℓ is 3-connected, then $|E(P_\ell)| \ge (\frac{|B_\ell|}{2(d-1)})^r + 1$, and

(iii) if L_ℓ is a cycle chain, then $|B_\ell| \ge 1$ and $|E(P_\ell)| \ge |B_\ell|$.

By choosing the notation of $\{c_{\ell-1}, d_{\ell-1}\}$, we may assume that $\bigcup_{i=2}^{\ell-1} P_i$ is between c_1 and $c_{\ell-1}$.

If L_ℓ is a cycle chain, then by Proposition 2.6 let P_ℓ be a path from x to $c_{\ell-1}$ in L_ℓ such that $B_\ell \subseteq V(P_\ell)$. Since $c_{\ell-1}, d_{\ell-1} \notin B_\ell$, we have $|E(P_\ell)| \ge |B_\ell|$. Note that $|B_\ell| = 1$ only if L_ℓ is a cycle and $B_\ell = \{x\}$.

If $L_\ell = K_4$, then we can find a path P_ℓ from x to $c_{\ell-1}$ in $L_\ell - d_{\ell-1}$ with $|E(P_\ell)| = 2 \ge (\frac{|L_\ell|}{2(d-1)})^r + 1 = (\frac{|B_\ell|}{2(d-1)})^r + 1$.

Now assume that L_ℓ is not a cycle chain and that $L_\ell \ne K_4$. Let $L_\ell' := L_\ell + \{xc_{\ell-1}, xd_{\ell-1}\}$. Then L_ℓ' is a 3-connected graph, the vertices $x, c_{\ell-1}, d_{\ell-1}$ have degree

at most $d+1$ in L'_ℓ, and all other vertices of L'_ℓ have degree at most d. So by Theorem 2.1(a), there is a cycle C_ℓ through $xc_{\ell-1}$ in $L'_\ell - d_{\ell-1}$ such that $|C_\ell| \geq (\frac{|L_\ell|}{2(d-1)})^r + 2$. Now $P_\ell := C_\ell - xc_{\ell-1}$ gives the desired path for (ii).

(4) Let $P := \bigcup_{i=1}^{\ell} P_i$. Clearly P is a path in $\bigcup_{i=1}^{\ell} L_i$ from x to a_0. We claim that $|E(P)| \geq (\sigma(\mathcal{L})+1)^r + 1$.

By (2), we have

$$|E(P)| \geq |E(P_1)| + \Big(\sum |B_i|\Big) + \left(\sum \left(\frac{|B_i|}{2(d-1)}\right)^r\right) + |E(P_\ell)|,$$

where the first summation is over all cycle chains L_i and the second is over all 3-connected L_i. Note that each B_i in the first summation can be written as $1^r + \cdots + 1^r$ ($|B_i|$ times), and this allows the application of Lemma 3.1 in the following argument.

If L_ℓ is 3-connected, then by (1) and (3),

$$\begin{aligned}|E(P)| &\geq (|B_1|+2)^r + \Big(\sum |B_i|\Big) + \left(\sum \left(\frac{|B_i|}{2(d-1)}\right)^r\right) + \left(\left(\frac{|B_\ell|}{2(d-1)}\right)^r + 1\right)\\ &\geq \left(2 + \sum_{i=1}^{\ell} |B_i|\right)^r + 1 \quad \text{(by Lemma 3.1)}\\ &> (\sigma(\mathcal{L})+1)^r + 1.\end{aligned}$$

If L_ℓ is a cycle chain, then by (1) and (3),

$$\begin{aligned}|E(P)| &\geq (|B_1|+2)^r + \Big(\sum |B_i|\Big) + \left(\sum \left(\frac{|B_i|}{2(d-1)}\right)^r\right) + (|B_\ell|-1) + 1\\ &\geq \left(1 + \sum_{i=1}^{\ell} |B_i|\right)^r + 1 \quad \text{(by Lemma 3.1)}\\ &= (\sigma(\mathcal{L})+1)^r + 1.\end{aligned}$$

(5) For $i = 1, \ldots, h-1$, we find paths Q_i from $\{a_{i-1}, b_{i-1}\}$ to $\{a_i, b_i\}$ in H_i, as in (2) of Case 1.

(6) We find a path Q_h in H_h as in (3) of Case 1.

(7) Let $C := (P \cup (\bigcup_{i=1}^{h} Q_i)) + \{y, xy, yy'\}$. Now C is a cycle in G through xy and

$$|C| \geq ((\sigma(\mathcal{L})+1)^r + 1) + \Big(\sum |A_i|\Big) + \left(\sum \left(\frac{|A_i|}{2(d-1)}\right)^r\right) + 2,$$

where the first summation is over all cycle chains H_i and the second is over all 3-connected H_i. Again, when we apply Lemma 3.1 in the following argument, each $|A_i|$ in the first summation may be written as $1^r + \cdots + 1^r$. Since $\sigma(\mathcal{L}) \geq |L_1| \geq |A_i|$ for all 3-connected H_i, we have

$$\begin{aligned}|C| &\geq \left(\sigma(\mathcal{L}) + 1 + (d-1)\sum_{i=1}^{h} |A_i|\right)^r + 3 \quad \text{(by Lemma 3.1)}\\ &= (\sigma(\mathcal{L}) + 1 + (d-1)\sigma(\mathcal{H}))^r + 3\\ &\geq n^r + 3 \quad \text{(by the observation preceding Case 1).}\end{aligned}$$

Case 3. $1 < t \le \ell$. Note that in this case there exist $y'' \in V(L_1) - \{c_1, d_1\}$ and a path Y in G from y to y'' disjoint from $(V(\mathcal{L}) - \{y''\}) \cup V(\mathcal{H})$. For convenience, let $S := (\bigcup_{i=2}^{t} B_i) - (B_1 \cup B_{t+1})$. We consider two subcases by comparing $\sigma(\mathcal{H})$ and $|S|$.

Subcase 3.1. $|S| < \sigma(\mathcal{H})$. Then $\mathcal{H} \neq \emptyset$ and $a_0 \neq b_0$. Since $\sum_{i=1}^{h} |A_i| \ge \sigma(\mathcal{H})$ and there are at most $d-2$ extreme 3-blocks of $G - y$ containing neither x nor y'', we have the following inequality:

$$\sigma(\mathcal{L}) - |S| + (d-1)\sum_{i=1}^{h} |A_i| \ge n - 1.$$

(1) First, we find a path P_1 from c_1 to d_1 in L_1 such that $|E(P_1)| \ge (|B_1|+1)^r + 1$.

If $L_1 = K_4$, then let P_1 be a Hamilton path from c_1 to d_1 in L_1. Hence $|E(P_1)| = 3 \ge (|L_1|+1)^r + 1 = (|B_1|+1)^r + 1$. If $L_1 \neq K_4$ then $5 \le |L_1| < n$. By Theorem 2.1(c), there is a cycle C_1 through c_1d_1 in L_1 such that $|C_1| \ge |L_1|^r + 3 = |B_1|^r + 3 > (|B_1|+1)^r + 2$. Then $P_1 := C_1 - c_1d_1$ gives the desired path.

(2) Next, we find $Q \subseteq (\bigcup_{i=2}^{t} L_i) - \{a_0b_0, c_td_t\}$ such that (1) $P_1 \cup Q$ is a path from $\{a_0, b_0\}$ to $\{c_t, d_t\}$ when $t \neq \ell$, and (2) $P_1 \cup Q$ is a path from $\{a_0, b_0\}$ to x when $t = \ell$.

Let $K := \bigcup_{i=2}^{t} L_i$. First, assume that $t \neq \ell$. If $\{a_0, b_0\} = \{c_t, d_t\}$, then by Lemma 3.5 we find a cycle D through a_0b_0 and c_1d_1 in K, and $Q := D - \{a_0b_0, c_1d_1\}$ is as desired. If $\{a_0, b_0\} \neq \{c_t, d_t\}$, then let K' be obtained from K by an H-transform at $\{a_0b_0, c_td_t\}$, and let a, c denote the new vertices. By Lemma 3.5, we find a cycle D' through ac and c_1d_1, and $Q := D' - \{a, c, c_1d_1\}$ is as desired.

Now assume that $t = \ell$. If $x \in \{a_0, b_0\}$, then by Lemma 3.5 there is a cycle D in K through a_0b_0 and c_1d_1, and $Q := D - \{a_0b_0, c_1d_1\}$ is as desired. So assume that $x \notin \{a_0, b_0\}$. Let K' be obtained from K by a T-transform at $\{x, a_0b_0\}$, and let a denote the new vertex. By Lemma 3.5, there is a cycle D in K' through xa and c_1d_1. Now $Q := D - \{a, c_1d_1\}$ is as desired.

We choose the notation of $\{a_0, b_0\}$ and $\{c_t, d_t\}$ so that $P_1 \cup Q$ is from a_0 to c_t.

(3) For each $t+1 \le i \le \ell-1$, we find a path P_i in L_i from $\{c_{i-1}, d_{i-1}\}$ to $\{c_i, d_i\}$ exactly as in (2) of Case 2 such that (i) $(\bigcup_{i=t+1}^{\ell-1} P_i) \cup P_1 \cup Q$ is a path from $\{c_{\ell-1}, d_{\ell-1}\}$ to $\{a_0, b_0\}$, (ii) $|E(P_i)| \ge (\frac{|B_i|}{2(d-1)})^r$ when L_i is 3-connected, and (iii) $|E(P_i)| \ge |B_i|+1$ when L_i is a cycle chain.

(4) If $t \neq \ell$, we find a path P_ℓ between $\{c_{\ell-1}, d_{\ell-1}\}$ and x exactly as in (3) of Case 2 such that (i) $(\bigcup_{i=t+1}^{\ell} P_i) \cup P_1 \cup Q$ is a path from $\{a_0, b_0\}$ to x, (ii) $|E(P_\ell)| \ge (\frac{|B_\ell|}{2(d-1)})^r + 1$ when L_ℓ is 3-connected, and (iii) $|E(P_\ell)| \ge |B_\ell|$ when L_ℓ is a cycle chain.

(5) For $i = 1, \ldots, h-1$, we find paths Q_i from $\{a_{i-1}, b_{i-1}\}$ to $\{a_i, b_i\}$ in $H_i - b_{i-1}$, as in (2) of Case 1, such that (i) $\bigcup_{i=1}^{h-1} Q_i$ is a path from a_0 to $\{a_{h-1}, b_{h-1}\}$, (ii) $|E(Q_i)| \ge (\frac{|A_i|}{2(d-1)})^r$ when H_i is 3-connected, and (iii) $|E(Q_i)| \ge |A_i|$ when H_i is a cycle chain.

(6) We find a path Q_h exactly as in (3) of Case 1 such that (i) $\bigcup_{i=1}^{h} Q_i$ is a path from a_0 to y', (ii) $|E(Q_h)| \ge (\frac{|A_h|}{2(d-1)})^r + 1$ when H_h is 3-connected, and (iii) $|E(Q_h)| \ge |A_h|$ when H_h is a cycle chain.

(7) Let $C := (P_1 \cup Q \cup (\bigcup_{i=t+1}^{\ell} P_i) \cup (\bigcup_{i=1}^{h} Q_i)) + \{y, xy, yy'\}$. Then C is a cycle in G through xy and, by (1)–(6), we have

$$|C| \ge (|L|+1)^r + \Big(\sum |B_i|\Big) + \Big(\sum \Big(\frac{|B_i|}{2(d-1)}\Big)^r\Big) + \Big(\sum |A_i|\Big) + \Big(\sum \Big(\frac{|A_i|}{2(d-1)}\Big)^r\Big) + 3,$$

where the first sum is taken over all cycle chains L_i for $t+1 \le i \le \ell$, the second is

over all 3-connected L_i for $t+1 \le i \le \ell$, the third is over all cycle chains H_i, and the fourth is over all 3-connected H_i. Because $|L| \ge |A_i|$ for all 3-connected H_i, and $|L| \ge |B_j|$ for all 3-connected L_j, we have

$$\begin{aligned}|C| &\ge \left(\sigma(\mathcal{L}) + 1 - |S| + (d-1)\sum_{i=1}^{h}|A_i|\right)^r + 3 \quad \text{(by Lemma 3.1)}\\ &\ge n^r + 3.\end{aligned}$$

The second inequality follows from the inequality in the first paragraph of this subcase.

Subcase 3.2. $|S| \ge \sigma(\mathcal{H})$. As in the previous subcase, we deduce the following inequality:

$$|B_1| + (d-1)\sum_{i=2}^{\ell}|B_i| \ge n-1.$$

(1) First, we find a path P_1 from y'' to $\{c_1, d_1\}$ in $L_1 - c_1d_1$ such that $|E(P_1)| \ge (|B_1|+1)^r + 1$.

Let L_1' denote the graph obtained from L_1 by a T-transform at $\{y'', c_1d_1\}$, and let y^* denote the new vertex. By Lemma 3.3 and since y'' has degree at most $d-1$ in L_1, L_1' is a 3-connected graph with maximum degree at most d. Since $5 \le |L_1'| < n$, Theorem 2.1 holds for L_1'. By Theorem 2.1(c), L_1' has a cycle C_1 through y^*y'' such that $|C_1| \ge |L_1'|^r + 3 = (|B_1|+1)^r + 3$. Then $P_1 := C_1 - y^*$ gives the desired path for (1).

We may choose the notation of $\{c_1, d_1\}$ so that P_1 is between y'' and c_1.

(2) For each $2 \le i \le \ell - 1$, we find a path P_i in L_i as in (2) of Case 2 such that (i) $\bigcup_{i=2}^{\ell-1} P_i$ is a path from c_1 to $\{a_{\ell-1}, b_{\ell-1}\}$, (ii) $|E(P_i)| \ge (\frac{|B_i|}{2(d-1)})^r$ when L_i is 3-connected, and (iii) $|E(P_i)| \ge |B_i| + 1$ when L_i is a cycle chain.

(3) We find a path P_ℓ as in (3) of Case 2 such that (i) $\bigcup_{i=2}^{\ell} P_i$ is a path from c_1 to x, (ii) $|E(P_\ell)| \ge (\frac{|B_\ell|}{2(d-1)})^r + 1$ when L_ℓ is 3-connected, and (iii) $|E(P_\ell)| \ge |B_\ell|$ when L_ℓ is a cycle chain.

(4) Let $C := (Y \cup (\bigcup_{i=1}^{\ell} P_i)) + xy$. Then C is a cycle in G through xy and

$$|C| \ge (|B_1|^r + 1) + \left(\sum |B_i|\right) + \left(\sum \left(\frac{|B_i|}{2(d-1)}\right)^r\right) + 2,$$

where the first sum is over all L_i which are cycle chains and the second is over all 3-connected L_i. Again, we may view $|B_i|$ in the first summation as $1^r + \cdots 1^r$ ($|B_i|$ times). Since $|B_1| \ge |B_i|$ for all 3-connected L_i, we have

$$\begin{aligned}|C| &\ge \left(|B_1| + 1 + (d-1)\sum_{i=2}^{\ell}|B_i|\right)^r + 3 \quad \text{(by Lemma 3.1)}\\ &\ge n^r + 3.\end{aligned}$$

The second inequality follows from the inequality in the first paragraph of this case. ∎

Next we show that the above proof gives rise to an $O(|E(G)|)$ algorithm which reduces Theorem 2.1(c) to (a), (b), and (c) of the same theorem for smaller graphs.

Algorithm Oneedge. Let n, d, r, G, e, be as in Lemma 6.1.

1. *Preprocessing.* Replace G by a 3-connected spanning subgraph of G with $O(|G|)$ edges. (This can be done in $O(|E(G)|)$ time using Lemma 3.4.)
2. Let $e = xy$. Decompose $G - y$ into 3-connected components. (This can be done in $O(|G|)$ time using Theorem 2.2.)
3. If there is only one 3-connected component of $G-y$, then $G-y$ is 3-connected. Let y' denote a neighbor of y other than x, and let $G' := (G - y) + xy'$ and $e' = xy'$. It suffices to find a cycle C' through e' in G' such that $|C'| \geq |G'|^r + 3$. So reduce Theorem 2.1(c) for G, e to Theorem 2.1(c) for G', e', with $|G'| < |G|$. (Note that this reduction takes constant time.)
4. Now assume that $G - y$ has at least two 3-connected components. If all 3-blocks of $G - y$ are cycles, find a cycle chain $\mathcal{I} = I_1 \dots I_i$ such that (i) $x \in V(I_1) - V(I_2)$, (ii) I_i is an extreme 3-block of $G - y$, and (iii) subject to (i) and (ii), $|V(\mathcal{I})|$ is maximum. (This can be done in $O(|G|)$ time by a simple search.) Then, find a neighbor $y' \in V(\mathcal{I}) - \{x\}$ of y, a Hamilton path P in $\mathcal{I}$ from x to y', and a path Q from y to y' disjoint from $V(\mathcal{I}) - \{y'\}$, so that $(P \cup Q) + \{y, xy, yy'\}$ gives the desired cycle. (These paths can be computed in $O(|G|)$ time as in the proof of Lemma 6.1.)
5. Now assume that $G - y$ has at least two 3-blocks, and at least one is 3-connected. We choose a 3-connected 3-block L of $G - y$ such that $|L|$ is maximum. Find the block chain $\mathcal{L} = L_1 \dots L_\ell$ such that $L_1 = L$ and $x \in V(L_\ell) - V(L_{\ell-1})$. Find a block chain $\mathcal{H} = H_1 \dots H_h$ in $G - y$ with $y' \in H_h$, and define a_i, b_i, A_i, c_j, d_j, B_j for $i = 1, \dots h$ and $j = 1, \dots, \ell$ as in the proof of Lemma 6.1. (All these can be done in $O(|G|)$ time by searching the 3-blocks of $G - y$.)
6. Suppose $\ell = 1$.
   - First, we need to find a path P in $\mathcal{L}$ from x to $\{a_0, b_0\}$ as in (1) of Case 1. We either find the desired P or reduce the problem of finding P to Theorem 2.1(c) for L_1, a_0b_0 or L_1', xx', both are smaller graphs. (From (1) of Case 1 in the proof of Lemma 6.1, this can be done in constant time.)
   - For each $1 \leq i \leq h - 1$, we want to find a path Q_i from $\{a_{i-1}, b_{i-1}\}$ to $\{a_i, b_i\}$ in H_i as in (2) of Case 1 in the proof of Lemma 6.1. We either find the desired Q_i or reduce the problem of finding Q_i to Theorem 2.1(a) for $H_i', a_{i-1}a_i, b_{i-1}$ or $H_i', a_{i-1}a, b_{i-1}$. (From (2) of Case 1 in the proof of Lemma 6.1, this can be done in $O(|H_i|)$ time.)
   - We need to find a path Q_h in H_h from a_{h-1} to y' as in (3) of Case 1. We either find the desired Q_h or reduce the problem of finding Q_h to Theorem 2.1(a) for $H_h', a_{h-1}y', b_{h-1}$. (From (3) of Case 1 in the proof of Lemma 6.1, this can be done in $O(|H_h|)$ time.)
   - Since $\sum_{i=1}^{h}(|H_i| - 2) = |V(\mathcal{H})| - 2$, we see that this step takes $O(|G|)$ time.
7. Suppose $1 = t < \ell$.
   - First, we need to find a path P_1 from $\{a_0, b_0\}$ to $\{c_1, d_1\}$ in L_1, as in (1) of Case 2 in the proof of Lemma 6.1. We either find the desired P_1 or reduce the problem of finding P_1 to Theorem 2.1(c) for either L_1, c_1d_1 or L_1', ac. (From (1) of Case 2 in the proof of Lemma 6.1, this can be done in constant time.)
     Assume that the notation is chosen so that P_1 is a path from a_0 to c_1.
   - For each $2 \leq i \leq \ell - 1$, we need to find a path P_i from $\{c_{i-1}, d_{i-1}\}$ to

$\{c_i, d_i\}$ in L_i as in (2) of Case 2 in the proof of Lemma 6.1. We either find the desired P_i or reduce the problem of finding P_i to Theorem 2.1(a) for either $L'_i, c_{i-1}c_i, d_{i-1}$ or $L'_i, c_{i-1}c, d_{i-1}$. (From (2) of Case 2 in the proof of Lemma 6.1, this can be done in $O(|L_i|)$ time.)

- We need to find a path P_ℓ from $\{c_{\ell-1}, d_{\ell-1}\}$ to x in L_ℓ, as in (3) of Case 2 in the proof of Lemma 6.1. We either find the desired path P_ℓ or reduce the problem of finding P_ℓ to Theorem 2.1(a) for $L'_\ell, xc_{\ell-1}, d_{\ell-1}$. (From (3) of Case 2 in the proof of Lemma 6.1, this can be done in $O(|L_\ell|)$ time.)
- Next we need to find paths Q_i, $1 \le i \le h$. This is taken care of exactly as in step 6 above.
  (Since $\sum_{i=1}^{h}(|H_i| - 2) = |V(\mathcal{H})| - 2$ and $\sum_{i=1}^{\ell}(|L_i| - 2) = |V(\mathcal{L})| - 2$, we see that all operations in this step can be done in $O(|G|)$ time.)

8. Suppose $1 < t \le \ell$. First, find a path Y from y to $y'' \in V(L_1) - V(L_2)$ such that $Y - y''$ is disjoint from $V(\mathcal{L}) \cup V(\mathcal{H})$. Define $S := (\bigcup_{i=2}^{t} B_i) - (B_1 \cup B_{t+1})$. (Note that Y and all B_i's can be found in $O(|G|)$ time, as in Case 3 in the proof of Lemma 6.1.)
9. Suppose $|S| < \sigma(\mathcal{H})$.
   - We need to find a path P_1 from c_1 to d_1 in L_1, as in (1) of Subcase 3.1 in the proof of Lemma 6.1. We either find the desired P_1 or reduce the problem of finding P_1 to Theorem 2.1(c) for L_1, c_1d_1. (From (1) of Subcase 3.1 in the proof of Lemma 6.1, this can be done in $O(|L_1|)$ time.)
   - Find $Q \subseteq (\bigcup_{i=2}^{t} L_i) - \{a_0b_0, c_td_t\}$ as in (2) of Subcase 3.1 in the proof of Lemma 6.1. (From (2) of Subcase 3.1 in the proof of Lemma 6.1, this can be done in $O(|G|)$ time.)
   - For each $t + 1 \le i \le \ell - 1$, we need to find a path P_i from $\{c_{i-1}, d_{i-1}\}$ to $\{c_i, d_i\}$, as in (3) of Subcase 3.1 in the proof of Lemma 6.1. (This can be done as in step 7 above, and hence in $O(|G|)$ time.)
   - Next, we need to find a path P_ℓ from $\{c_{\ell-1}, d_{\ell-1}\}$ to x in L_ℓ as in (4) of Subcase 3.1 in the proof of Lemma 6.1. (This can be done as in step 7 above, and hence in $O(|L_\ell|)$ time.)
   - For each $1 \le i \le h - 1$, we want to find a path Q_i as in (5) of Subcase 3.1 in the proof of Lemma 6.1. (This can be done as in step 6 above, and hence in $O(|H_i|)$ time.)
   - Finally, we find a Q_h in H_h from $\{a_{h-1}, b_{h-1}\}$ to y' as in (6) of Subcase 3.1 in the proof of Lemma 6.1. (This can be done as in step 6 above, and hence in $O(|H_h|)$ time.)
     (Since $\sum_{i=1}^{h}(|H_i| - 2) = |V(\mathcal{H})| - 2$ and $\sum_{i=1}^{\ell}(|L_i| - 2) = |V(\mathcal{L})| - 2$, we see that all operations in this step can be done in $O(|G|)$ time.)
10. Suppose $|S| \ge \sigma(\mathcal{H})$.
    - First, we need to find a path P_1 from y'' to $\{c_1, d_1\}$ in L_1, as in (1) of Subcase 3.2 in the proof of Lemma 6.1. We either find the desired P_1 or reduce the problem of finding P_1 to Theorem 2.1(c) for L'_1, y^*y''. (From (1) of Subcase 3.2 in the proof of Lemma 6.1, this can be done in $O(|L_1|)$ time.)
    - For each $2 \le i \le \ell - 1$, we need to find a P_i as in (2) of Subcase 3.2 in the proof of Lemma 6.1. (This can be done as in step 7 above for each i, and hence, in $O(|L_i|)$ time.)

- Next, we want to find a path P_ℓ from $\{c_{\ell-1}, d_{\ell-1}\}$ to x in L_ℓ as in (3) of Subcase 3.2 in the proof of Lemma 6.1. (This can be done as in step 7 above, and hence, in $O(|L_\ell|)$ time.)
  (All operations in this step can be done in $O(|G|)$ time since $\sum_{i=1}^{\ell}(|L_i|-2) = |V(\mathcal{L})| - 2$.)

We summarize the above procedure as follows.

PROPOSITION 6.2. *Given G, e, n, d, r as in Lemma 6.1, we can, in $O(|E(G)|)$ time, either*

(1) *find a cycle C through e in G such that $|C| \geq |G|^r + 3$, or*
(2) *reduce Theorem 2.1(c) for G, e to (a), (b), or (c) of the same theorem for smaller 3-connected graphs.*

Moreover, any smaller graph in (2) results from a 3-connected 3-block of $G - y$ which is not K_4. Hence, any smaller graph in (2) contains a vertex that does not belong to any other smaller graph in (2).

7. Conclusions. We now complete the proof of Theorem 2.1. Let n, d, r, G be given as in the theorem. We will prove the conclusions by applying induction on n. When $n = 5$, then G is isomorphic to one of the following three graphs: K_5, K_5 minus an edge, or the wheel on five vertices. In each case, we can verify that Theorem 2.1 holds. So assume that $n \geq 6$ and that Theorem 2.1 holds for all 3-connected graphs with at most $n-1$ vertices. Then (a) holds by Lemma 4.1, (b) holds by Lemma 5.1, and (c) holds by Lemma 6.1. This completes the proof of Theorem 2.1. □

ALGORITHM CYCLE. Let G be a 3-connected graph with maximum degree at most d, let $e = xy \in E(G)$, and assume $|G| \geq 5$. The following procedure finds a cycle C through e in G with $|C| \geq |G|^r + 3$.

1. *Preprocessing.* Replace G with a 3-connected spanning subgraph of G with $O(|G|)$ edges.
2. Apply Algorithm Oneedge to G, e. We either find the desired cycle C or we reduce the problem to Theorem 2.1(a), (b), or (c) for some 3-connected graphs G_i, for which $|G_i| < |G|$ and each G_i contains a vertex which does not belong to any other G_i.
3. Replace each G_i with a 3-connected spanning subgraph of G_i with $O(|G_i|)$ edges.
4. Apply Algorithm Avoidvertex to those G_i for which Theorem 2.1(a) needs to be applied. Apply Algorithm Twoedge to those G_i for which Theorem 2.1(b) needs to be applied. Apply Algorithm Oneedge to those G_i for which Theorem 2.1(c) needs to be applied.
5. Repeat steps 3 and 4 for new 3-connected graphs.
6. In the final output, replace all virtual edges by paths in G to complete the desired cycle C.

Note that step 1 takes $O(|E(G)|)$ time by Lemma 3.4 and step 2 takes $O(|E(G)|)$ time by Proposition 4.2.

By Lemma 3.4, step 3 spends $O(|E(G_i)|)$ time for each G_i from step 2. Note that each G_i in step 2 contains a vertex which does not belong to any other G_i. By Theorem 2.2 and since each G_i contributes at most three additional edges due to T-transform or H-transform, the total number of edges in step 2 is at most $3|E(G)| - 6 + 3|V(G)|$. Hence step 3 takes $O(|E(G)|)$ time.

From Propositions 4.2, 5.2, and 6.2, we see that step 4 spends $O(|E(G_i)|)$ time for each G_i from step 2. By Theorem 2.2 and since each G_i contributes at most three additional edges due to T-transform or H-transform, the total number of edges in step

4 is at most $\sum_i(3|E(G_i)| - 6 + 3|V(G_i)|)$. Since each G_i in step 2 contains a vertex which does not belong to any other G_i, step 4 takes $O(|G|^2)$ time.

Since there are at most $|G|$ iterations, we see that Algorithm Cycle takes $O(|G|^3)$ time.

REFERENCES

[1] A. Björklund and T. Husfeldt, *Finding a path of superlogarithmic length*, in Proceedings of the 29th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 2380, Springer, Berlin, 2002, pp. 985–992.

[2] J. A. Bondy and M. Simonovits, *Longest cycles in* 3*-connected cubic graphs*, Canad. J. Math., 32 (1980), pp. 987–992.

[3] G. Chen and X. Yu, *Long cycles in* 3*-connected graphs*, J. Combin. Theory Ser. B, 86 (2002), pp. 80–99.

[4] N. Chiba and T. Nishizeki, *The Hamiltonian cycle problem is linear-time solvable for* 4*-connected planar graphs*, J. Algorithms, 10 (1989), pp. 187–211.

[5] F. Chung, private communication.

[6] T. Feder, R. Motwani, and C. Subi, *Approximating the longest cycle problem in sparse graphs*, SIAM J. Computing, 31 (2002), pp. 1596–1607.

[7] Z. Gao and X. Yu, *Convex programming and circumference of* 3*-connected graphs of low genus*, J. Combin. Theory Ser. B, 69 (1997), pp. 39–51.

[8] B. Grünbaum and H. Walther, *Shortness exponents of families of graphs*, J. Combin. Theory Ser. A, 14 (1973), pp. 364–385.

[9] D. A. Holton and B. D. McKay, *The smallest non-Hamiltonian* 3*-connected cubic planar graphs have* 38 *vertices*, J. Combin. Theory Ser. B, 45 (1988), pp. 305–319.

[10] J. E. Hopcroft and R. E. Tarjan, *Dividing a graph into triconnected components*, SIAM J. Comput., 2, (1973), pp. 135–158.

[11] T. Ibaraki and H. Nagamochi, *A linear-time algorithm for finding a sparse* k*-connected spanning subgraph of a* k*-connected graph*, Algorithmica, 7 (1992), pp. 583–596.

[12] B. Jackson, *Longest cycles in* 3*-connected cubic graphs*, J. Combin. Theory Ser. B, 41 (1986), pp. 17–26.

[13] B. Jackson and N. Wormald, *Longest cycles in* 3*-connected planar graphs*, J. Combin. Theory Ser. B, 54 (1992), pp. 291–321.

[14] B. Jackson and N. C. Wormald, *Longest cycles in* 3*-connected graphs of bounded maximum degree*, in Graphs, Matrices, and Designs, R. S. Rees, ed., Marcel Dekker, New York, 1993, pp. 237–254.

[15] D. Karger, R. Motwani, and G. D. S. Ramkumar, *On approximating the longest path in a graph*, Algorithmica, 18 (1997), pp. 82–98.

[16] J. Malkevitch, *Polytopal graphs*, in Selected Topics in Graph Theory, Vol. 3, L. Beineke and R. Wilson, eds., Academic Press, New York, 1988, pp. 169–188.

[17] J. W. Moon and L. Moser, *Simple paths on polyhedra*, Pacific J. Math., 13 (1963), pp. 629–631.

[18] W. T. Tutte, *A theorem on planar graphs*, Trans. Amer. Math. Soc., 82 (1956), pp. 99–116.

[19] W. T. Tutte, *Connectivity in Graphs*, University of Toronto Press, Toronto, ON, 1966.

[20] S. Vishwanathan, *An approximation algorithm for finding a long path in Hamiltonian graphs*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, SIAM, Philadelphia, pp. 680–685.

[21] H. Whitney, *A theorem on graphs*, Ann. of Math., 32 (1931), pp. 378–390.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1171–1200

A SWITCHING LEMMA FOR SMALL RESTRICTIONS AND LOWER BOUNDS FOR k-DNF RESOLUTION*

NATHAN SEGERLIND†, SAM BUSS‡, AND RUSSELL IMPAGLIAZZO§

Abstract. We prove a new switching lemma that works for restrictions that set only a small fraction of the variables and is applicable to formulas in disjunctive normal form (DNFs) with small terms. We use this to prove lower bounds for the Res(k) propositional proof system, an extension of resolution which works with k-DNFs instead of clauses. We also obtain an exponential separation between depth d circuits of bottom fan-in k and depth d circuits of bottom fan-in $k+1$.

Our results for Res(k) are as follows:

1. The $2n$ to n weak pigeonhole principle requires exponential size to refute in Res(k) for $k \leq \sqrt{\log n / \log\log n}$.
2. For each constant k, there exists a constant $w > k$ so that random w-CNFs require exponential size to refute in Res(k).
3. For each constant k, there are sets of clauses which have polynomial size Res($k+1$) refutations but which require exponential size Res(k) refutations.

Key words. propositional proof complexity, Boolean circuit complexity, switching lemmas, lower bounds, k-DNFs, resolution, Res(k), circuit bottom fan-in, random restriction, Sipser functions, weak pigeonhole principles, random CNFs

AMS subject classifications. 03F20, 68Q15

DOI. 10.1137/S0097539703428555

1. Introduction.

This paper studies the complexity of Res(k), a propositional refutation system that extends resolution by allowing k-DNFs instead of clauses [24]. The complexity of propositional proof systems has close connections to open problems in computational and circuit complexity [14, 23, 29, 6], as well as implications for the run times of satisfiability algorithms and automated theorem provers. Resolution is one of the most studied proof systems and is used as the basis for many satisfiability algorithms. Backtracking algorithms for satisfiability, such as the popular Davis–Putnam–Logeman–Loveland procedure (DPLL), that branch on a single variable provide tree-like resolution refutations on unsatisfiable formulas. General resolution proofs correspond to adding a limited form of memoization (previously refuted subproblems are saved for reuse rather than refuted again) to DPLL. Res(k) corresponds to algorithms that branch on more general conditions: the value of any function of up to k variables.

The Res(k) systems are also interesting as intermediates between previously studied proof systems. Resolution can be thought of as Res(1), and depth two Frege can be thought of as Res(n) (where n is the number of variables). The Res(k) systems provide a transition between these systems. Moreover, statements provable in the theory

*Received by the editors May 28, 2003; accepted for publication (in revised form) February 24, 2004; published electronically July 20, 2004.
http://www.siam.org/journals/sicomp/33-5/42855.html

†School of Mathematics, Institute for Advanced Study, Einstein Drive, Princeton, NJ 08540 (nsegerli@math.ias.edu). The research of this author was supported in part by NSF grants DMS-0100589 and CCR-0098197.

‡Department of Mathematics, University of California, San Diego, La Jolla, CA 92092-0112 (sbuss@ucsd.edu). The work of this author was supported in part by NSF grant DMS-0100589.

§Department of Computer Science, University of California, San Diego, La Jolla, CA 92092-0112 (russell@cs.ucsd.edu). The work of this author was supported in part by NSF grant CCR-0098197 and USA–Israel BSF grant 97-00188.

$T_2^2(\alpha)$ (a fragment of Peano's arithmetic that allows induction only on Σ_2^b predicates) correspond to propositional statements with quasi-polynomial size Res(polylog(n)) refutations [24]. T_2^2 is the weakest fragment of Peano's arithmetic known to be able to use counting arguments such as the weak pigeonhole principle [25]. On the other hand, these counting tautologies are known to require exponential-size resolution refutations. Thus, there must be a critical range for k between 1 and polylog(n) where these arguments become possible in subexponential size. More generally, we can ask, When does increasing k give the Res(k) system more power? Is there a reason to want to branch on more complex functions in satisfiability algorithms? Does such branching give algorithms better performance in the average case?

We give partial answers to all of these questions. In particular we prove the following:

1. The $2n$ to n weak pigeonhole principle requires size $2^{\Omega(n^\epsilon)}$ to refute in Res(k) for all $k \leq \sqrt{\log n / \log\log n}$. So having large bottom fan-in is necessary for counting arguments.
2. For each k, there exists a constant $w > k$ so that random w-CNFs require size $2^{\Omega(n^\epsilon)}$ to refute in Res(k). Therefore, extending DPLL algorithms to branch on multiple (but a constant number of) variables will not make run times subexponential on average.
3. Res($k+1$) is exponentially more powerful than Res(k). We demonstrate sets of clauses that have polynomial size Res($k+1$) refutations but require size $2^{\Omega(n^\epsilon)}$ to refute in Res(k). Therefore, increasing the complexity of branching conditions can make proofs exponentially smaller.

Our lower bounds are proved using a new kind of switching lemma. A switching lemma provides conditions under which an OR of small ANDs can be rewritten as an AND of small ORs after the application of a random restriction [1, 18, 21, 4]. Our switching lemma differs from previous switching lemmas in that the random restriction is allowed to set a small number of the variables, even as few as $n^{1-\epsilon}$ out of n. The trade-off is that ORs of *extremely* small ANDs are transformed into ANDs of modestly small ORs. Therefore, our switching lemma cannot be iterated to prove lower bounds for proof systems of depth more than two. However, one application of our switching lemma suffices to prove lower bounds for the Res(k) proof systems, because each line in such a proof is of depth two with small bottom fan-in.

Our switching lemma also gives an exponential separation between depth d circuits with bottom fan-in k from depth d circuits with bottom fan-in $k+1$. This refines a previous result of Håstad [21], which states that for all d there exist $\epsilon > \delta > 0$ so that there are functions on n variables, computable with polynomial size, depth d circuits of bottom fan-in n^ϵ but which require exponential size to compute with depth d circuits of bottom fan-in n^δ. Our result also refines results of Cai, Chen, and Håstad [12]. They showed that for each constant d, there exist functions computable with polynomial-size, depth $d+1$, bottom fan-in 2 circuits that require exponential size to compute with depth d circuits, and that for each constant k, there exists a function of n variables computable by depth d circuits of polynomial-size and bottom fan-in $O(\log n)$ that requires exponential size to compute with depth d circuits of bottom fan-in k.

Because resolution may be viewed as Res(1), our results for Res(k) generalize known results for resolution. The weak pigeonhole principle (for any number of pigeons) is known to require an exponential number of steps to refute in resolution [35, 20, 36, 11, 5, 15, 28, 30, 31], and we generalize these lower bounds for the case

of the cn to n pigeonhole principle. Resolution refutations of randomly chosen sets of clauses are also known to require exponential size [13, 5, 8]. We extend these results to general Res(k) systems, although as k increases, so does the width of the random CNFs for which our lower bounds apply.

Our work also extends previous research on the Res(k) system. The complexity of Res(k) refutations was first studied by Krajíček [24], who was motivated by the connection between Res(polylog(n)) and the provability of combinatorial statements in the arithmetic theory $T_2^2(\alpha)$. Atserias, Bonet, and Esteban [3] gave exponential lower bounds for Res(2) refutations of the $2n$ to n weak pigeonhole principle and of random 3-CNFs. They also proved a quasi-polynomial separation between Res(2) and resolution; this separation was later strengthened to almost-exponential by Atserias and Bonet [2]. Esteban, Galesi, and Messner [17] showed that that there is an exponential separation between tree-like Res(k) and tree-like Res($k+1$), and that general (DAG-like) Res(k) requires high *space* to refute random CNFs and the weak pigeonhole principle.

The lower bounds for Res(k) refutations of the weak pigeonhole principle given by Atserias, Bonet, and Esteban [3] apply only for $k = 2$; our lower bound works for nonconstant k up to $\sqrt{\log n / \log\log n}$. On the other hand, Maciel, Pitassi, and Woods [25] gave quasi-polynomial-size refutations in Res(polylog(n)). Our results show that superconstant bottom fan-in is necessary for subexponential-size proofs of the weak pigeonhole principle. Indeed, after the preliminary version of this paper appeared [33], our techniques were extended by Alexander Razborov to prove that the weak pigeonhole principle requires exponential size to refute in Res($\epsilon \log n / \log\log n$) [32].

Our lower bounds for Res(k) refutations of random w-CNFs are the first such lower bounds for Res(k) with $k \geq 3$. Atserias, Bonet, and Esteban [3] gave exponential lower bounds for random 3-CNFs in Res(2). We extend these results to Res(k), although the width w increases with k (it is $4k^2 + 2$). At present, the Res(k) systems are the strongest fragments of bounded-depth Frege systems for which we know there are superpolynomial-size lower bounds for refutations of random sets of clauses.

The separation between Res($k+1$) from Res(k) is the first for $k \geq 2$. The earlier work of Atserias and Bonet [2] gave a $2^{\Omega(2^{\log^{\epsilon} n})}$ separation of Res(2) from Res(1), and our result improves this to $2^{\Omega(n^{\epsilon})}$.

In the time since the preliminary version of this paper appeared [33], we have extended one of our results. The original separation of Res($k+1$) from Res(k) was based on clauses of width $O(\log n)$, whereas the new separation uses clauses of constant width. We include proofs for both results.

1.1. Outline of the paper. Background material, including the basics of the Res(k) proof system, is given in section 2. In section 3 we prove the switching lemma. Section 4 applies the switching lemma to prove a separation between constant-depth circuits of bottom fan-in $k+1$ and constant-depth circuits of bottom fan-in k. In section 5 we prove that Res(k) refutations of sets of clauses whose lines are represented by short decision trees can be transformed into narrow resolution refutations. This conversion is used in combination with the switching lemma to prove lower bounds for Res(k) refutations. Lower bounds for Res(k) refutations of the weak pigeonhole principle are proved in section 6 and lower bounds for Res(k) refutations of random CNFs are proved in section 7. The separations between Res($k+1$) and Res(k) are proved in sections 8 and 9.

2. Background. A *literal* is a variable or its negation. A *term* is a constant 0 or 1 or a conjunction of literals. Our convention is that a term is specified as a set of literals, with 1 corresponding to the empty set and 0 to any literal and its negation. We say that a term T contains a literal l if $l \in T$, and that a term T contains a variable x if either $x \in T$ or $\neg x \in T$. We will often identify literals with terms of size one, and will write l instead of $\{l\}$. A *DNF* is a disjunction of terms, specified as a set of terms. A *k-DNF* is a DNF whose terms are each of size at most k. A *clause* is a 1-DNF, i.e., a disjunction of literals. The width of a clause C, written $w(C)$, is the number of literals appearing in C. The width of a set of clauses is the maximum width of any clause in the set. A *CNF* is a conjunction of clauses, specified as a set of clauses. A *k-CNF* is a CNF whose clauses are each of width at most k. Two terms t and t' are *consistent* if there is no literal l with $l \in t$ and $\neg l \in t'$.

A *restriction* ρ is a map from a set of variables to $\{0, 1, *\}$. For a formula F, *the restriction of F by ρ*, $F \restriction_\rho$ is defined as usual, simplifying only when a subexpression has become explicitly constant. For any restriction ρ, let $\mathrm{dom}(\rho)$ denote the set of variables to which ρ assigns the value 0 or 1.

Resolution is a refutation system for propositional logic. The input to a resolution refutation is a set of clauses $\mathcal{C}$; a resolution refutation consists of a derivation of the empty clause from the clauses in $\mathcal{C}$ using only the resolution inference: $\frac{A \vee x \quad \neg x \vee B}{A \vee B}$. Notice that every line in a resolution refutation is a clause. $w_R(\mathcal{C})$ denotes the minimum width of a resolution refutation of $\mathcal{C}$; if $\mathcal{C}$ is satisfiable, then there is no refutation and we use the convention that $w_R(\mathcal{C})$ is ∞.

The $\mathrm{Res}(k)$ refutation system is a generalization of resolution that can reason using k-DNFs.

DEFINITION 2.1. *Res(k) is the refutation system whose lines are k-DNFs and whose inference rules are given below (A, B are k-DNF's, $1 \leq j \leq k$, and $l, l_1, \ldots, l_j$ are literals):*

$$\textit{Subsumption:} \quad \frac{A}{A \vee l} \qquad \textit{AND-introduction:} \quad \frac{A \vee l_1 \ \cdots \ A \vee l_j}{A \vee \bigwedge_{i=1}^{j} l_i}$$

$$\textit{Cut:} \quad \frac{A \vee \bigwedge_{i=1}^{j} l_i \quad B \vee \bigvee_{i=1}^{j} \neg l_i}{A \vee B} \qquad \textit{AND-elimination:} \quad \frac{A \vee \bigwedge_{i=1}^{j} l_i}{A \vee l_i}$$

Let $\mathcal{C}$ be a set of k-DNFs. A $\mathrm{Res}(k)$ derivation from $\mathcal{C}$ *is a sequence of k-DNFs $F_1, \ldots, F_m$ so that each F_i either belongs to $\mathcal{C}$ or follows from the preceding lines by an application of one of the inference rules. For a set of k-DNFs $\mathcal{C}$, a* $\mathrm{Res}(k)$ refutation of $\mathcal{C}$ *is a derivation from $\mathcal{C}$ whose final line is the empty clause. The size of a Res(k) refutation is the number of lines it contains. $S_k(\mathcal{C})$ denotes the minimum size of a Res(k) refutation of $\mathcal{C}$. If $\mathcal{C}$ is satisfiable, then $\mathcal{C}$ has no refutation and we use the convention that $S_k(\mathcal{C})$ is ∞.*

We do not use the exact definition of the $\mathrm{Res}(k)$ system in our arguments; the main property we use is *strong soundness*: if F is inferred from $F_1, \ldots, F_j$, and $t_1, \ldots, t_j$ are mutually consistent terms of $F_1, \ldots, F_j$, respectively, then there is a term t of F implied by $\bigwedge_{i=1}^{j} t_i$. In other words, any reason why $F_1, \ldots, F_k$ are true implies a reason why F is true. This is stronger than mere soundness.

LEMMA 2.2. *Res(k) is strongly sound.*

We also use the following well-known interpolation property for resolution.

LEMMA 2.3. *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be unsatisfiable sets of clauses on disjoint sets of variables. If there is a resolution refutation Γ of $\mathcal{C}_1 \cup \mathcal{C}_2$, then there is a refutation Γ'*

of either $\mathcal{C}_1$ or $\mathcal{C}_2$. Moreover, $w(\Gamma') \leq w(\Gamma)$.

2.1. The Chernoff bounds. Throughout this paper we will repeatedly make use of a simplified form of the Chernoff bounds. These formulations come from standard references on applying such bounds in algorithmics; cf. [26, 37].

LEMMA 2.4. *Let $X_1, \ldots, X_n$ be independent random indicator variables. Let $\mu = E[\sum_{i=1}^{n} X_i]$.*

$\Pr\left[\sum_{i=1}^{n} X_i < \frac{\mu}{2}\right] \leq e^{-\mu/8}$ and $\Pr\left[\sum_{i=1}^{n} X_i > 2\mu\right] \leq e^{-\mu/4}$.

2.2. Miscellaneous notation. We will use the notation $[k] = \{i \mid 1 \leq i \leq k\}$. For graphs $G = (V, E)$ and $S \subseteq V$ we will write $G - S$ to denote the induced subgraph on $V \setminus S$.

3. The switching lemma. A switching lemma is a guarantee that after the application of a randomly chosen restriction, a disjunction of small ANDs can be represented by a conjunction of small ORs, thus "switching" an OR into an AND. We use a slightly stronger variation: after the application of a random restriction, a k-DNF can be represented by a short decision tree.

DEFINITION 3.1. *A* decision tree *is a rooted binary tree in which every internal node is labeled with a variable, the edges leaving a node correspond to whether the variable is set to 0 or 1, and the leaves are labeled with either 0 or 1. Every path from the root to a leaf may be viewed as a partial assignment. For a decision tree T and $v \in \{0, 1\}$, we write the set of paths (partial assignments) that lead from the root to a leaf labeled v as $\mathrm{Br}_v(T)$. For a partial assignment ρ, $T \restriction_\rho$ is the decision tree obtained by deleting from T every edge whose label conflicts with ρ and contracting along each edge whose label belongs to ρ. We say that a decision tree T* strongly represents *a DNF F if for every $\pi \in \mathrm{Br}_0(T)$, for all $t \in F$, $t \restriction_\pi = 0$ and for every $\pi \in \mathrm{Br}_1(T)$, there exists $t \in F$, $t \restriction_\pi = 1$. The representation height of F, $h(F)$, is the minimum height of a decision tree strongly representing F.*

Notice that the function computed by a decision tree of height h can be computed both by an h-CNF and by an h-DNF.

Our switching lemma will exploit a trade-off based on the minimum size of a set of variables that meets each term of a k-DNF. When this quantity is small, we can build a decision tree by querying these variables and recursing on the $(k-1)$-DNFs created. When this quantity is large, the DNF has many disjoint terms and is likely to be satisfied by a random restriction.

DEFINITION 3.2. *Let F be a DNF, and let S be a set of variables. If every term of F contains a variable from S, then we say that S* is a cover of F. *The* covering number of F, $c(F)$, *is the minimum cardinality of a cover of F.*

For example, the 3-DNF $xyz \vee \neg x \vee yw$ has covering number two.

The switching lemma is shown to hold for all distributions which satisfy certain properties. When we apply the switching lemma, we will show that the random restrictions used satisfy these properties.

THEOREM 3.3. *Let $k \geq 1$, let $s_0, \ldots, s_{k-1}$ and $p_1, \ldots, p_k$ be sequences of positive numbers, and let $\mathcal{D}$ be a distribution on partial assignments so that for every $i \leq k$ and every i-DNF G, if $c(G) > s_{i-1}$, then $\Pr_{\rho \in \mathcal{D}}[G \restriction_\rho \neq 1] \leq p_i$. Then for every k-DNF F,*

$$\Pr_{\rho \in \mathcal{D}}\left[h(F \restriction_\rho) > \sum_{i=0}^{k-1} s_i\right] \leq \sum_{i=1}^{k} 2^{\left(\sum_{j=i}^{k-1} s_j\right)} p_i.$$

Proof. We proceed by induction on k. First consider $k = 1$. If $c(F) \leq s_0$, then at most s_0 variables appear in F. We can construct a height $\leq s_0$ decision tree that strongly represents $F \restriction_\rho$ by querying all of the variables of $F \restriction_\rho$. On the other hand, if $c(F) > s_0$, then $\Pr_{\rho\in\mathcal{D}}[F \restriction_\rho \neq 1] \leq p_1$. Therefore, $h(F \restriction_\rho)$ is nonzero with probability at most $p_1 2^{\sum_{j=1}^{k-1} s_j} = p_1$ (because $k = 1$).

For the induction step, assume that the theorem holds for all k-DNFs, let F be a $(k+1)$-DNF, and let $s_0, \ldots, s_k$ and $p_1, \ldots, p_{k+1}$ be sequences of positive numbers satisfying the hypotheses of the theorem. If $c(F) > s_k$, then by the conditions of the lemma, $\Pr_{\rho\in\mathcal{D}}[F \restriction_\rho \neq 1] \leq p_{k+1}$. Because

$$p_{k+1} \leq \sum_{i=1}^{k+1} 2^{\sum_{j=i}^{k} s_j} p_i,$$

we have that $h(F \restriction_\rho)$ is nonzero with probability at most $\sum_{i=1}^{k+1} 2^{\sum_{j=i}^{k} s_j} p_i$.

Consider the case when $c(F) \leq s_k$. Let S be a cover of F of size at most s_k. Let π be any assignment to the variables in S. Because each term of F contains at least one variable from S, $F \restriction_\pi$ is a k-DNF. By combining the induction hypothesis with the union bound, we have that

$$\begin{aligned}\Pr_{\rho\in\mathcal{D}}\left[\exists \pi \in \{0,1\}^S \;\; h((F \restriction_\rho) \restriction_\pi) > \sum_{i=0}^{k-1} s_i\right] &\leq 2^{s_k}\left(\sum_{i=1}^{k} 2^{\left(\sum_{j=i}^{k-1} s_j\right)} p_i\right) \\ &< \sum_{i=1}^{k+1} 2^{\left(\sum_{j=i}^{k} s_j\right)} p_i.\end{aligned}$$

In the event that for all $\pi \in \{0,1\}^S$, $h((F \restriction_\rho) \restriction_\pi) \leq \sum_{i=0}^{k-1} s_i$, we construct a decision tree for $F \restriction_\rho$ as follows. First, query all variables in S unset by ρ, and then underneath each branch, β, simulate a decision tree of minimum height strongly representing $(F \restriction_\rho) \restriction_\beta$. For each such β, let $\pi = (\rho \cup \beta) \restriction_S$, and note that $h((F \restriction_\rho) \restriction_\beta) = h((F \restriction_\rho) \restriction_\pi)$. Therefore the height of the resulting decision tree is at most $s_k + \max_{\pi\in\{0,1\}^S} h((F \restriction_\rho) \restriction_\pi) \leq \sum_{i=0}^{k} s_i$.

Now we show that the decision tree constructed above strongly represents $F \restriction_\rho$. Let π be a branch of the tree. Notice that $\pi = \beta \cup \sigma$, where β is an assignment to the variables in $S \setminus \mathrm{dom}(\rho)$ and σ is a branch of a tree that strongly represents $(F \restriction_\rho) \restriction_\beta$. Consider the case that π leads to a leaf labeled 1. In this case, σ satisfies a term t' of $(F \restriction_\rho) \restriction_\beta$. We may choose a term t of F so that $t' = (t \restriction_{\rho\cup\beta})$, and $\pi = \beta \cup \sigma$ satisfies the term $t \restriction_\rho$ of $F \restriction_\rho$. Now consider the case that π leads to a leaf labeled 0. There are two cases, $(F \restriction_\rho) \restriction_\beta = 0$ and $(F \restriction_\rho) \restriction_\beta \neq 0$. If $(F \restriction_\rho) \restriction_\beta = 0$, then for every term t of $F \restriction_\rho$, t is inconsistent with β and hence with π. If $(F \restriction_\rho) \restriction_\beta \neq 0$, then because the subtree underneath β strongly represents $(F \restriction_\rho) \restriction_\beta$, for every term t of $(F \restriction_\rho) \restriction_\beta$, t is inconsistent with σ. Therefore, every term of $F \restriction_\rho$ is inconsistent with either β or σ, and thus with $\pi = \beta \cup \sigma$. □

We usually use this theorem in the following normal form for the parameters.

COROLLARY 3.4. *Let k, s, and d be positive integers, let γ and δ be real numbers from the range $(0, 1]$, and let $\mathcal{D}$ be a distribution on partial assignments so that for every k-DNF G, $\Pr_{\rho\in\mathcal{D}}[G \restriction_\rho \neq 1] \leq d2^{-\delta(c(G))^\gamma}$. For every k-DNF F,*

$$\Pr_{\rho\in\mathcal{D}}[h(F \restriction_\rho) > 2s] \leq dk2^{-\delta' s^{\gamma'}},$$

where $\delta' = 2(\delta/4)^k$ *and* $\gamma' = \gamma^k$.

Proof. Let $s_i = (\delta/4)^i(s^{\gamma^i})$ and $p_i = d2^{-4s_i}$. Note that $s_{i-1}/4 \geq (\delta/4)s_{i-1} = (\delta/4)(\delta/4)^{i-1}s^{\gamma^{i-1}} \geq (\delta/4)^i s^{\gamma^i} = s_i$. It follows that $\sum_{j=i}^{k} s_j \leq \sum_{j\geq i} s_i/4^{j-i} \leq 2s_i$. Also, for any i-DNF G, with $c(G) \geq s_{i-1}$, $\Pr_{\rho\in\mathcal{D}}[G\restriction_\rho \neq 1] \leq d2^{-\delta(c(G))^\gamma} \leq d2^{-\delta s_{i-1}^\gamma} = 2^{-\delta(\delta/4)^{i-1}(s^{\gamma^{i-1}})^\gamma} = d2^{-4s_i}$. Thus, we can apply Theorem 3.3 with parameters $p_1,\ldots,p_k,s_0,\ldots,s_{k-1}$. For every k-DNF F,

$$\Pr_{\rho\in\mathcal{D}}[h(F\restriction_\rho) > 2s] \leq \Pr_{\rho\in\mathcal{D}}\left[h(F\restriction_\rho) > \sum_{i=0}^{k-1} s_i\right] \leq \sum_{i=1}^{k} 2^{\left(\sum_{j=i}^{k-1} s_j\right)} p_i$$

$$\leq \sum_{i=1}^{k} 2^{2s_i}(d2^{-4s_i}) \leq dk2^{-2s_k} = dk2^{-\delta' s^{\gamma'}}. \qquad \square$$

3.1. Switching with small restrictions. In this subsection, we show that small, uniform restrictions meet the conditions for the switching lemma. Using Corollary 3.4, k-DNFs can then be converted into decision trees—*using restrictions that set only a polynomially small fraction of the bits*. We include it here for comparison with previous switching lemmas. Later, it will be used to prove the lower bound on Res(k) refutations of random CNFs. More complicated distributions are used for our other results.

DEFINITION 3.5. *Let* $n > 0$ *and* $p \in [0,1]$. *Define* $\mathcal{D}_p$ *to be the family of random restrictions which arises by assigning variables* $*$ *with probability* $1-p$, *and* 0, 1 *each with probability* $\frac{p}{2}$.

LEMMA 3.6. *Let* $i \geq 1$, *let* G *be an* i*-DNF, and let* ρ *be chosen from* $\mathcal{D}_p$. *Then* $\Pr[G\restriction_\rho \neq 1] \leq e^{-\frac{c(G)p^i}{i2^i}}$.

Proof. Because every covering set of G has size at least $c(G)$, there is a set of variable-disjoint terms of size at least $c(G)/i$ (such a set can be found by greedily choosing a maximal set of disjoint terms). Each of these variable-disjoint terms is satisfied with independent probability at least $(p/2)^i$. Therefore,

$$\Pr_{\rho\in\mathcal{D}_p}[G\restriction_\rho \neq 1] \leq \left(1-\left(\frac{p}{2}\right)^i\right)^{\frac{c(G)}{i}} \leq e^{-\left(\frac{p}{2}\right)^i\frac{c(G)}{i}} = e^{-\frac{c(G)p^i}{i2^i}}. \qquad \square$$

Combining this with the switching lemma shows that a k-DNF is strongly represented by a short decision tree when restricted.

COROLLARY 3.7. *Let* $k \geq 1$ *be given. There exists* $\gamma > 0$ *so that for any* k*-DNF* F, $w > 0$, $p \geq n^{-1/(2k^2)}$, $\Pr_{\rho\in\mathcal{D}_p}[h(F\restriction_\rho) > w] \leq k2^{-\gamma w n^{-1/2}}$.

Proof. In the notation of Corollary 3.4, set $p = n^{-1/2k^2}$, $d = 1$, $\gamma = 1$, $s = w/2$, and $\delta = (\log e)\frac{p^k}{k2^k} = (\log e)\frac{n^{-1/2k}}{k2^k}$. Combining Lemma 3.6 with Corollary 3.4 shows that for every k-DNF F,

$$\Pr_{\rho\in\mathcal{D}_p}[h(F\restriction_\rho) > w] \leq k2^{-2(w/2)(\delta^k/4^k)} = k2^{-w(\log e)^k n^{-1/2}/4^k k^k 2^{k^2}}.$$

Because k is fixed, we may take $\gamma = (\log e)^k/4^k k^k 2^{k^2}$. $\square$

4. An application to circuit bottom fan-in. Our first application of the switching lemma is an exponential-size separation between depth d circuits of bottom fan-in k and depth d circuits of bottom fan-in $k+1$.

Our circuits are organized into alternating layers of AND and OR gates, with connections appearing only between adjacent levels. NOT gates may have only variables as their inputs. The output gate is said to be at level one, the gates feeding into the output gate are said to be at level two, and so forth. The depth of a circuit is the maximum depth of an AND or OR gate in the circuit. The size of a circuit is the number of AND and OR gates appearing in it. The *bottom fan-in* of a depth d circuit is the maximum number of inputs of a gate at level d. For more detail on the basics of constant depth circuits, consult the survey by Boppana and Sipser [10].

4.1. The functions.

DEFINITION 4.1 (see [34, 10]). *Let integers d and $m_1, \ldots, m_d$ be given, and let there be variables $x_{i_1,\ldots,i_d}$ for $1 \leq i_j \leq m_j$.*

$$f_d^{m_1,\ldots,m_d} = \bigwedge_{i_1 \leq m_1} \bigvee_{i_2 \leq m_2} \cdots \bigodot_{i_d \leq m_d} x_{i_1,\ldots,i_d},$$

where $\bigodot = \bigvee$ if d is even, and $\bigodot = \bigwedge$ if d is odd.

The Sipser function f_d^m *is $f_d^{m_1,\ldots,m_d}$, with $m_1 = \sqrt{m/logm}$, $m_2 = \cdots = m_{d-1} = m$, and $m_d = \sqrt{dm \log m/2}$.*

The modified Sipser function $g_d^{m,k}$ *is $f_{d+1}^{m_1,\ldots,m_d,k}$, with $m_1 = \sqrt{m/logm}$, $m_2 = \cdots = m_{d-1} = m$, and $m_d = 4\sqrt{dm \log m/2}$.*

Notice that the function f_d^m depends on $m^{d-1}\sqrt{d/2}$ many variables and can be computed by a circuit of depth d and size linear in the number of variables. Furthermore, we will often identify these functions with the circuits defining them.

Our result builds upon the earlier result that it is impossible to decrease the bottom fan-in of a circuit computing a Sipser function without increasing the size or the depth. Moreover, an OR of depth d, small bottom fan-in circuits requires exponential size to compute f_d^m.

THEOREM 4.2 (see [21]). *For all $d \geq 1$, there exists $\epsilon_d > 0$ so that if a depth d, bottom fan-in k circuit with an AND gate at the output and at most S gates in levels 1 through $d-1$ computes f_d^m, then either $k \geq m^{\epsilon_d}$ or $S \geq 2^{m^{\epsilon_d}}$.*

For all $d \geq 1$, there exists $\beta_d > 0$ so that if a depth $d+1$, bottom fan-in k circuit with an OR gate at the output and at most S gates in levels 1 through d computes f_d^m, then either $S \geq 2^{m^{\beta_d}}$ or $k \geq m^{\beta_d}$.

We use the modified Sipser function $g_d^{m,k+1}$ to obtain the exponential separation between depth $d+1$, bottom fan-in $k+1$ and depth $d+1$, bottom fan-in k circuits. For each $i_1, \ldots, i_d$, we say that the variables $x_{i_1,\ldots,i_d,1}, \ldots, x_{i_1,\ldots,i_d,k}$ come from *block* $(i_1, \ldots, i_d)$. Variables in the same block occur in the same bottom-level conjunction of $g_d^{m,k}$. Notice that the function $g_d^{m,k}$ has $4m^{d-1}\sqrt{d/2}$ many blocks and $4km^{d-1}\sqrt{d/2}$ many variables. Moreover, it can be computed by a circuit of depth $d+1$, bottom fan-in k and size linear in the number of variables.

4.2. The lower bounds. We will show that depth $d+1$ circuits with bottom fan-in k require exponential size to compute $g_d^{m,k+1}$. In light of Theorem 4.2, it suffices to consider only circuits with an AND gate at the output level. Furthermore, we consider only the case when d is even. This ensures that all gates at depth d are OR gates. The case for odd d is dual and we simply invert the random restriction used. Each gate at depth d computes a k-DNF, and we will apply random restrictions which almost certainly collapse all of the k-DNFs to narrow CNFs and thus collapse the circuits to depth d circuits with small bottom fan-in. On the other hand, the

random restrictions will probably leave $g_d^{m,k+1}$ containing f_d^m as a subfunction, and thus we obtain a contradiction to Theorem 4.2.

DEFINITION 4.3. *Let m, d, and k be given. Set $m_1 = \sqrt{m/logm}$, $m_2 = \cdots = m_{d-1} = m$, and $m_d = 4\sqrt{dm \log m/2}$.*

Let $\mathcal{B}_{d,0}^{m,k+1}$ be the random distribution on partial assignments given by the following experiment: For each $i_1 \leq m_1, \ldots, i_d \leq m_d$, with independent probability $\frac{1}{2}$ either set $x_{i_1,\ldots,i_d,j} = *$, for all $j \in [k+1]$, or uniformly choose a $0/1$ assignment to $\{x_{i_1,\ldots,i_d,j} \mid j \in [k+1]\}$ which sets at least one of the variables to 0. The dual distribution, $\mathcal{B}_{d,1}^{m,k+1}$, selects a restriction according to $\mathcal{B}_{d,0}^{m,k+1}$ and then inverts the 0's and 1's.

LEMMA 4.4. *Let $k \geq 1$ be given. There exists a constant $\gamma_k > 0$ so that for every k-DNF F,*

$$\Pr_{\rho \in \mathcal{B}_{d,0}^{m,k+1}} [F \restriction_\rho \neq 1] \leq 2^{-\gamma_k c(F)}.$$

Proof. We say that two terms T and T' are *block-disjoint* if no variable of T shares a block with a variable of T'. More formally, whenever a variable $x_{i_1,\ldots,i_{d+1}}$ appears in T and a variable $x_{j_1,\ldots,j_{d+1}}$ appears in T', we have that $(i_1, \ldots, i_d) \neq (j_1, \ldots, j_d)$. Because each term involves at most k variables, there must be a set of $c(F)/k$ many variable-disjoint terms, and hence a set of $c(F)/(k(k+1))$ many block-disjoint terms.

We now show that each term is satisfied with probability at least $\frac{1}{6^k}$. Because the literals of a term come from at most k distinct blocks, the chance that every variable in the term is set to 0 or 1 is at least $\frac{1}{2^k}$. Conditioned on this event, the probability of satisfying the term is at least $\frac{1}{3^k}$. To see this, consider the chance of satisfying each literal of the term in turn, conditioned on the event of satisfying the previous literals. If a variable from that block has already been set to 0, then clearly the probability of satisfying the current literal is $\frac{1}{2}$. If not, then suppose there are l variables in the block of the current variable that have not yet been set to a value. The probability of satisfying the current literal is at least $(2^{l-1} - 1)/(2^l - 1)$. Because there are $k+1$ variables and the term has size at most k, $l \geq 2$, and thus the probability is at least $\frac{1}{3}$.

The events of satisfying block-disjoint terms are independent; therefore we have

$$\Pr_{\rho \in \mathcal{B}_{d,0}^{m,k+1}} [F \restriction_\rho \neq 1] \leq \left(1 - \frac{1}{6^k}\right)^{c(F)/(k(k+1))}.$$

Set $\gamma_k = -\log_2(1 - \frac{1}{6^k})/(k(k+1))$. □

Symmetrically, the dual result holds for k-CNFs when we apply a random restriction from $\mathcal{B}_{d,1}^{m,k+1}$.

LEMMA 4.5. *Let $k \geq 1$ be given. There exists a constant ϵ_k so that for all d, for all w sufficiently large with respect to k, and for every depth $d+1$, bottom fan-in k circuit C of size $S \leq 2^{\epsilon_k w}$, when ρ is chosen from $\mathcal{B}_{d,0}^{m,k+1}$ ($\mathcal{B}_{d,1}^{m,k+1}$), with probability at least $3/4$, $C \restriction_\rho$ is equivalent to a depth d, bottom fan-in w circuit with at most S gates in levels 1 through $d-1$.*

Proof. We will solve for the particular values of ϵ_k and w after going through the calculations.

We consider the case when d is even; the case when d is odd is handled by using the restrictions $\mathcal{B}_{d,1}^{m,k+1}$ instead of $\mathcal{B}_{d,0}^{m,k+1}$. Each gate at depth d is an OR gate and its inputs are AND gates of fan-in at most k. For each gate g at depth d, we let F_g denote the k-DNF computed by the subcircuit at g.

Suppose that there is a partial assignment $\rho \in \mathcal{B}^{m,k}_{d,0}$ so that for each depth d gate g of C, $h(F_g \restriction_\rho) < w$. For each g at depth d, let T_g be the shortest decision tree representing $F_g \restriction_\rho$. We can compute $C \restriction_\rho$ with a depth d, bottom fan-in w circuit with at most S gates in levels 1 through $d-1$ by starting with C, replacing each level d gate g with the conjunction of the negated branches of $\mathrm{Br}_0(T_g)$, and then merging these conjuncts with the AND gate at depth $d-1$ to which g sends its output.

We now show that for ρ chosen according to the distribution $\mathcal{B}^{m,k}_{d,0}$, with probability at least $\frac{3}{4}$, every depth d gate g of C has $h(F_g \restriction_\rho) < w$.

Let g be a depth d gate of the circuit. By combining Lemma 4.4 with Corollary 3.4, setting $d = 1$, $\gamma = 1$, $s = w/2$, and $\delta = \gamma_k$ shows that

$$\Pr_{\rho \in \mathcal{B}^{m,k}_{d,0}} [h(F_g) > w] \leq k 2^{-w\gamma_k^k/4^k}.$$

Because there are at most $S = 2^{\epsilon_k w}$ many gates at depth d, by the union bound, there exists a gate with $h(F_g) > w$ with probability at most $2^{w(\epsilon_k - \gamma_k^k/4^k) + \log k}$. We simply take ϵ_k sufficiently small so that this probability is less than $1/4$. □

THEOREM 4.6. *For all $k \geq 1$, $d \geq 1$, there exists $\epsilon_k, \epsilon_d > 0$ so that for every m sufficiently large, every size S, depth $d+1$, bottom fan-in k circuit for $g_d^{m,k+1}$ has $S \geq 2^{\epsilon_k m^{\epsilon_d}}$.*

Proof. We will have to take m sufficiently large to apply Theorem 4.2 and Lemma 4.5, and large enough for an application of the Chernoff bounds. Set $w = m^{\epsilon_d}$ (with ϵ_d from Theorem 4.2) and $S = 2^{\epsilon_k w}$ (with ϵ_k from Lemma 4.5). Furthermore, we consider the case when d is even; the case when d is odd is handled by using the restrictions $\mathcal{B}^{m,k+1}_{d,1}$ instead of $\mathcal{B}^{m,k+1}_{d,0}$.

Suppose, for the sake of contradiction, that C is a size S, depth d, bottom fan-in k circuit computing $g_d^{m,k+1}$.

Fix an OR gate at depth d in $g_d^{m,k+1}$. When ρ is chosen from the distribution $\mathcal{B}^{m,k+1}_{d,0}$, the expected number of blocks underneath this gate that are left unset is $2\sqrt{dm \log m/2}$. By the Chernoff bounds, with probability at most $e^{-\sqrt{dm \log m/2}/4}$ there are fewer than $\sqrt{dm \log m/2}$ blocks left unset by ρ underneath this gate.

Because there are $m^{d-3/2}/\sqrt{\log m}$ many depth d gates in $g_d^{m,k+1}$, by the union bound, the probability that there exists a depth d gate underneath which there are fewer than $\sqrt{dm \log m/2}$ many blocks unset is at most $(m^{d-3/2}/\sqrt{\log m}) e^{-\sqrt{dm \log m/2}/4}$. This tends to 0 as m tends to infinity.

On the other hand, by Lemma 4.5, with probability at least $3/4$, $C \restriction_\rho$ is equivalent to a depth d, bottom fan-in w circuit with at most S gates in levels 1 through $d-1$.

Therefore we may choose $\rho \in \mathcal{B}^{m,k+1}_{0,d}$ so that underneath each depth d gate of $g_d^{m,k+1}$ there are at least $\sqrt{dm \log m/2}$ many blocks unset by ρ, and $C \restriction_\rho$ is equivalent to a depth d, bottom fan-in w circuit with $\leq S$ gates in levels $1, \ldots, d-1$.

Because $C \restriction_\rho$ computes $g_d^{m,k+1} \restriction_\rho$, a restriction of it computes f_d^m: set some blocks to 0 and collapse the other blocks to a single variable. This gives a depth d circuit with $\leq S$ gates in levels $1, \ldots, d-1$, and bottom fan-in w that computes f_d^m, a contradiction to Theorem 4.2. □

5. Decision trees and Res(k) refutations. All of our lower bounds for Res(k) refutations use the fact that when the lines of a Res(k) refutation can be strongly represented by short decision trees, the Res(k) refutation can be converted into a narrow resolution refutation.

THEOREM 5.1. *Let $\mathcal{C}$ be a set of clauses of width $\leq h$. If $\mathcal{C}$ has a Res(k) refutation so that for each line F of the refutation, $h(F) \leq h$, then $w_R(\mathcal{C}) \leq kh$.*

Proof. We will use the short decision trees to construct a narrow refutation of $\mathcal{C}$ in resolution augmented with subsumption inferences: whenever $A \subseteq B$, infer B from A. These new inferences simplify our proof, but they may be removed from the resolution refutation without increasing the size or the width.

For a line F of the Res(k) refutation, let T_F be a decision tree of minimum height that strongly represents F. Notice that for each initial clause $C \in \mathcal{C}$, T_C is the tree that queries the (at most h) variables in C, stopping with a 1 if the clause becomes satisfied and stopping with a 0 if the clause becomes falsified.

For any partial assignment π let C_π be the clause of width $\leq h$ that contains the negation of every literal in π, i.e., the clause that says that branch π was not taken. We construct a resolution refutation of width $\leq kh$ by deriving C_π for each line F of the refutation and each $\pi \in \mathrm{Br}_0(T_F)$.

Notice that for $\pi \in \mathrm{Br}_0(T_\emptyset)$, $C_\pi = \emptyset$, and for each $C \in \mathcal{C}$, for the unique $\pi \in \mathrm{Br}_0(T_C)$, $C_\pi = C$.

Let F be a line of the refutation that is inferred from previously derived formulas $F_1, \ldots, F_j$, $j \leq k$. Assume we have derived all $C_\pi \in \mathrm{Br}_0(T_{F_i})$ for $1 \leq i \leq j$. To guide the derivation of $\{C_\pi \mid \pi \in \mathrm{Br}_0(T_F)\}$, we construct a decision tree that represents $\bigwedge_{i=1}^{j} F_i$. The tree (call it T) begins by simulating T_{F_1} and outputting 0 on any 0-branch of T_{F_1}. On any 1-branch, it then simulates T_{F_2}, etc. If all j branches are 1, T outputs 1; otherwise T outputs 0. The height of T is at most $jh \leq kh$, so the width of any such C_π, with $\pi \in \mathrm{Br}(T)$, is at most kh. The set of clauses $\{C_\sigma \mid \sigma \in \mathrm{Br}_0(T)\}$ can be derived from the previously derived clauses by subsumption inferences because every $\sigma \in \mathrm{Br}_0(T)$ contains some $\pi \in \bigcup_{i=1}^{j} \mathrm{Br}_0(T_{F_i})$.

We now show that for every $\sigma \in \mathrm{Br}_1(T)$, there exists a $t \in F$ so that σ satisfies t. Choose $\pi_1 \in \mathrm{Br}_1(T_{F_1}), \ldots, \pi_j \in \mathrm{Br}_1(T_{F_j})$ so that $\pi_1 \cup \cdots \cup \pi_j = \sigma$. Because the decision trees $T_{F_1}, \ldots, T_{F_j}$ strongly represent the k-DNFs $F_1, \ldots, F_j$, there exist terms $t_1 \in F_1, \ldots, t_j \in F_j$ so that $\bigwedge_{i=1}^{j} t_i$ is satisfied by σ. By strong soundness of Res(k), there exists $t \in F$ so that σ satisfies t.

Let $\sigma \in \mathrm{Br}_0(T_F)$ be given. Because T_F strongly represents F, σ falsifies all terms of F. By the preceding paragraph, for all $\pi \in \mathrm{Br}(T)$, if π is consistent with σ, then $\pi \in \mathrm{Br}_0(T)$ (otherwise, σ would not falsify the term of F satisfied by π). For each node v in T, let π_v be the path (viewed as a partial assignment) from the root to v. Bottom-up, from the leaves to the root, we recursively derive $C_{\pi_v} \vee C_\sigma$ for each v so that π_v is consistent with σ. When we reach the root, we will have derived C_σ. If v is a leaf, then $\pi_v \in \mathrm{Br}_0(T)$ so it has already been derived. If v is labeled with a variable that appears in σ, call it x, then there is a child u of v with $\pi_u = \pi_v \cup \{x\}$. Therefore, $C_{\pi_v} \vee C_\sigma = C_{\pi_u} \vee C_\sigma$. By induction, the clause $C_{\pi_u} \vee C_\sigma$ has already been derived. If v is labeled with a variable x that does not appear in σ, then for both of the children of v, call them v_1, v_2, the paths π_{v_1} and π_{v_2} are consistent with σ. Moreover, $C_{\pi_{v_1}} \vee C_\sigma = x \vee C_{\pi_v} \vee C_\sigma$ and $C_{\pi_{v_2}} \vee C_\sigma = \neg x \vee C_{\pi_v} \vee C_\sigma$. Resolving these two previously derived clauses gives us $C_{\pi_v} \vee C_\sigma$. ∎

We will use this theorem after we apply a random restriction which simultaneously collapses every line of a Res(k) refutation to a short decision tree. Hence, we can use a width lower bound for resolution refutations of a restricted tautology to give a size lower bound for Res(k) refutations of the original tautology.

COROLLARY 5.2. *Let $\mathcal{C}$ be a set of clauses of width $\leq h$, let Γ be a Res(k) refutation of $\mathcal{C}$, and let ρ be a partial assignment so that for every line F of Γ,*

$h(F\restriction_\rho) \leq h$. Then $w_R(\mathcal{C}\restriction_\rho) \leq kh$.

6. Lower bounds for the weak pigeonhole principle.

DEFINITION 6.1. *The m to n* pigeonhole principle, *PHP_n^m, is the following set of clauses:*

1. *For each $i \in [m]$, $\bigvee_{j\in[n]} x_{i,j}$.*
2. *For each $i, i' \in [m]$ with $i \neq i'$, $\neg x_{i,j} \vee \neg x_{i',j}$.*

THEOREM 6.2. *For every $c > 1$, there exists $\epsilon > 0$ so that for all n sufficiently large, if $k \leq \sqrt{\log n / \log\log n}$, then every Res($k$) refutation of PHP_n^{cn} has size at least 2^{n^ϵ}.*

The idea of the proof for Theorem 6.2 is as follows: Suppose there is a small Res(k) refutation of the weak pigeonhole principle. Then by applying a random restriction we obtain a low-width resolution refutation of the restricted pigeonhole principle. By the well-known lower bounds on the width of resolution refutations of the pigeonhole principle, this is impossible.

In order to make the random restriction method work, we prove lower bounds for the pigeonhole principle restricted to a low degree graph. Because these principles reduce to the pigeonhole principle by setting some variables to 0, this suffices to prove lower bounds for the pigeonhole principle. The difficulty with applying random restrictions directly to the clauses of the pigeonhole principle is that there are clauses of high width which are not satisfied with very high probability. If we were to choose a random subset of the holes and place into each hole a randomly chosen pigeon, then a clause of the form $\bigvee_{i=1}^m x_{i,j}$ would be satisfied with probability no better than the chance that hole j is in the random subset (this will be no better than a constant in our proof). At the heart of this problem is that each hole j appears in cn distinct variables, $x_{1,j}, \ldots, x_{cn,j}$, and restricting the principle to low-degree graphs solves this.

DEFINITION 6.3. *Let $G = (U \cup V, E)$ be a bipartite graph. The* pigeonhole principle of G, PHP(G), *is the following set of clauses:*

1. *For each $u \in U$*

$$\bigvee_{\substack{v\in V\\ \{u,v\}\in E}} x_{u,v}.$$

2. *For each $u, u' \in [m]$, with $u \neq u'$, and each $v \in V$ with $\{u, v\} \in E$ and $\{u', v\} \in E$,*

$$\neg x_{u,v} \vee \neg x_{u',v}.$$

DEFINITION 6.4. *Let $G = (U \cup V, E)$ be a bipartite graph. The* maximum degree of G, $\Delta(G)$, *is defined to be* $\max_{v\in V} \deg v$.

Furthermore, we assume that all Res(k) refutations have been put into a normal form in which no term of any DNF asks that two pigeons be mapped to the same hole. See, for example, [3].

DEFINITION 6.5. *Let $G = (U \cup V, E)$ be a bipartite graph. A term is said to be in* pigeon-normal-form *if it does not contain two literals $x_{u,v}$ and $x_{u',v}$ with $u \neq u'$. A DNF is said to be in pigeon-normal-form if all of its terms are in pigeon-normal-form and a Res(k) refutation is said to be in pigeon-normal-form if every line is in pigeon-normal-form.*

Every Res(k) refutation of $PHP(G)$ can be transformed into a refutation in pigeon-normal-form which at most doubles the number of lines in the proof. When

there is an AND-introduction inference that creates a line not in pigeon-normal-form, say

$$\frac{(A \vee x_{u,v}) \quad (A \vee x_{u',v}) \quad \cdots \quad (A \vee l_j)}{A \vee \left(x_{u,v} \wedge x_{u',v} \wedge \bigwedge_{i=3}^{j} l_i \right)},$$

replace the inference by a derivation that cuts $A \vee x_{u',v}$ with $\neg x_{u,v} \vee \neg x_{u',v}$ to obtain $A \vee \neg x_{u,v}$. Cut this with $A \vee x_{u,v}$ to obtain A. We may proceed through the rest of the proof with A because it subsumes $A \vee x_{u,v} \wedge x_{u',v} \wedge \bigwedge_{i=3}^{j} l_i$.

6.1. Random restrictions.

DEFINITION 6.6. *For a bipartite graph $G = (U \cup V, E)$ and a real number $p \in [0,1]$, let $\mathcal{M}_p(G)$ denote the distribution on partial assignments which arises from the following experiment:*

Independently, for each $v \in V$, with probability $1-p$ choose to match v, and with probability p leave v unmatched. If v is matched, uniformly select a neighbor u of v, set $x_{u,v}$ to 1, and for every $w \neq u$ that is a neighbor of v, set $x_{w,v}$ to 0. Moreover, for each $v' \neq v$, set $x_{u,v'} = 0$.

Let V_ρ be the set of vertices of V matched by ρ, let U_ρ be the set of vertices of U matched by ρ, and let $S_\rho = U_\rho \cup V_\rho$.

These restrictions randomly associate pigeons with holes in an injective way. While some pigeons can be associated with multiple holes, no two pigeons can be associated with the same hole. It is easy to check that for any $\rho \in \mathcal{M}_p(G)$, we have that $PHP(G) \restriction_\rho = PHP(G - S_\rho)$.

LEMMA 6.7. *Let $p \in [0,1]$, $i \in [k]$ be given. Let $G = (U \cup V, E)$ be a bipartite graph with $\Delta = \Delta(G)$. Let F be an i-DNF in pigeon-normal-form.*

$$\Pr_{\rho \in \mathcal{M}_p(G)} [F \restriction_\rho \neq 1] \leq 2^{-\frac{(\log e)(1-p)^i c(F)}{i\Delta^{i+1}}}.$$

Proof. For a term T, define the *holes of* T as $\mathrm{Holes}(T) = \{v \mid x_{u,v} \in T \text{ or } \neg x_{u,v} \in T\}$. We say that two terms T and T' are *hole-disjoint* if $\mathrm{Holes}(T) \cap \mathrm{Holes}(T') = \emptyset$.

Because F contains at least $c(F)/i$ many variable-disjoint terms, and each hole $v \in V$ appears in at most Δ many variables, F must contain at least $c(F)/i\Delta$ many hole-disjoint terms.

The events of satisfying hole-disjoint terms are independent, and for a given term, T, the probability that $T \restriction_\rho = 1$ is at least $(1-p)^i/\Delta^i$. This is because with probability $(1-p)^i$, every hole of T is matched, and with probability at least $1/\Delta^i$ the holes are matched in a way that satisfies T (here we use the fact that F is in pigeon-normal-form). Therefore, we have the following inequalities:

$$\Pr_\rho [F \restriction_\rho \neq 1] \leq \left(1 - (1-p)^i/\Delta^i\right)^{\frac{c(F)}{i\Delta}} \leq \left(e^{-(1-p)^i/\Delta^i}\right)^{\frac{c(F)}{i\Delta}} = 2^{-\frac{(\log e)(1-p)^i c(F)}{i\Delta^{i+1}}}. \qquad \square$$

6.2. Width lower bounds for resolution. For the lower bound proof to work, we need a graph G so that after the application of a random restriction ρ, with high probability, $PHP(G) \restriction_\rho$ requires high width to refute in resolution. We call such graphs *robust*, and in this subsection we probabilistically demonstrate robust, low-degree graphs.

DEFINITION 6.8. *A bipartite graph G is said to be* (p, w)-robust *if, when ρ is selected from $\mathcal{M}_p(G)$, with probability at least $\frac{1}{2}$, $w_R(PHP(G) \restriction_\rho) \geq w$.*

All we need for the size lower bound is the following lemma, which is proven probabilistically in section 6.2.1. Readers who believe that random graphs should be robust can skip to the proof of the lower bound.

LEMMA 6.9. *For all $c > 1$, there exists $d > 0$, so that for n sufficiently large, there exists a $(3/4, n/24)$-robust graph with $\Delta(G) \leq d \log n$ on the vertex sets $[cn]$ and $[n]$.*

6.2.1. Existence of robust graphs. As a starting point, we use a now standard lower bound of $w_R(PHP(G))$ in terms of the expansion of G.

DEFINITION 6.10. *For a vertex $u \in U$, let $N(u)$ be its set of neighbors. For a subset $V' \subseteq V$, let its boundary be*

$$\partial V' = \{u \in U \mid |N(u) \cap V'| = 1\}.$$

A bipartite graph G is an (m, n, r, f)-expander if $|V| = m$, $|U| = n$, and for all $V' \subseteq V$, $|V'| \leq r$, $|\partial V'| \geq f|V'|$.

THEOREM 6.11 (see [8]). *If G is a bipartite graph that is an (m, n, r, f)-expander, then $w_R(PHP(G)) \geq rf/2$.*

DEFINITION 6.12. *Let $G_{m,n,p}$ be the distribution on bipartite graphs with vertex sets $[m]$ and $[n]$ in which every edge is included with independent probability p.*

The following lemma was proven by Atserias, Bonet and Esteban [3].

LEMMA 6.13 (see [3]). *Let $m = cn$, $q = \frac{48c \ln m}{m}$, $\alpha = \frac{1}{mq}$, and $f = \frac{nq}{6}$. Let G be selected according to the distribution $G_{m,n,q}$.*

$$\Pr_G [G \text{ is an } (m, n, \alpha m, f)\text{-expander }] \geq \frac{2}{3}.$$

LEMMA 6.14. *Let $m = cn$, let $q \geq \frac{48c \ln m}{m}$, and let G be selected according to the distribution $G_{m,n,q}$.*

$$\Pr_G [w_R(PHP(G)) \geq n/12] \geq \frac{2}{3}.$$

Proof. Let $\alpha = \frac{1}{mq}$ and $f = \frac{nq}{6}$. Because $\alpha m f/2 = (1/mq)m(nq/6)/2 = n/12$, an application of Theorem 6.11 shows that when G is selected according to $G_{m,n,\frac{48c \ln m}{m}}$, with probability at least $\frac{2}{3}$, $w_R(PHP(G)) \geq n/12$.

Now consider G selected according to $G_{m,n,q}$, with $q \geq \frac{48c \ln m}{m}$. Whenever G_0 is an edge-induced subgraph of G_1, $w_R(PHP(G_1)) \geq w_R(PHP(G_0))$ because a refutation of $PHP(G_1)$ can always be transformed into a refutation of $PHP(G_0)$ by setting some variables to 0. Therefore, by increasing the probability of including an edge, the probability of having no small resolution refutation for $PHP(G)$ only increases. ∎

We now prove Lemma 6.9.

Proof. Set $m = cn$, $p = \frac{3}{4}$, and $q = \frac{192c \ln m}{m}$. Consider the joint distribution that arises by selecting G according to $G_{m,n,q}$ and ρ according to $\mathcal{M}_{3/4}(G)$. We will bound the probability that the degree is too large, that too many holes are matched, and that the restricted graph is expanding.

By the Chernoff bounds, for each $v \in [n]$ the probability that v has degree in excess of $2mq$ is at most $e^{-mq/4}$. By the union bound, the probability that there exists some $v \in [n]$ of degree in excess of $2mq$ is at most $ne^{-mq/4}$. Similarly, the probability that there exists some $v \in [m]$ of degree in excess of $2mq$ is at most $me^{-nq/4}$. Therefore, the probability that the maximum degree of G exceeds $2mq$ is bounded as follows:

$$ne^{-mq/4} + me^{-nq/4} = ne^{-m192c \ln m/m} + me^{-n192c \ln m/m} = O(n^{-191}).$$

Remember that V_ρ is the set of holes matched by the restriction ρ. By the Chernoff bounds, the probability that $|V_\rho| \geq 2n(1-p) = n/2$ is at most $e^{-\frac{n(1-p)}{4}} = e^{-n/16}$.

We now bound the probability that $G - S_\rho$ is an expander. First, up to renaming vertices, $G - S_\rho$ is distributed as $G_{m_\rho,n_\rho,q}$, with $n_\rho = n - |V_\rho|$ and $m_\rho = m - |U_\rho|$. This is because for fixed sets of vertices $V_0 \subseteq V$ and $U_0 \subseteq U$, when we condition on the event that $V_\rho = V_0$ and $U_\rho = U_0$, the edges $\{u, v\}$ with $u \in U \setminus U_0$ and $v \in V \setminus V_0$ are included in $G - S_\rho$ with independent probability q. Now, condition on the event that $|V_\rho| \leq n/2$. We have that $m_\rho = m - |U_\rho| \geq m - n/2 \geq m/2$ and thus $q = 192c \ln m/m \geq 192c \ln m_\rho/m \geq 192c \ln m_\rho/2m_\rho = 48 \cdot 2c \ln m_\rho/m_\rho$. Because $\frac{m_\rho}{n_\rho} \leq \frac{cn}{n/2} = 2c$, we can apply Lemma 6.14 and deduce that $w_R(PHP(G - S_\rho)) \geq n_\rho/12$ with probability at least $\frac{2}{3}$. Because $n_\rho \geq n/2$, with the same probability, $w_R(G - S_\rho) \geq n/24$.

Combining the three inequalities from the preceding paragraphs shows that the probability that G contains a vertex of degree in excess of $192c \ln m$, that V_ρ contains more than $n/2$ vertices, or that $w_R(PHP(G - S_\rho)) < n/24$ is at most

$$\frac{1}{3} + O(n^{-191}) + e^{-n/16}.$$

For sufficiently large n, this probability is bounded above by $\frac{1}{2}$. By averaging over the choices of the edges, there exists a bipartite graph G on vertex sets $[cn]$ and $[n]$ with $\Delta(G) \leq 2mq = 384c \ln(cn)$, so that upon selection of $\rho \in \mathcal{R}_{3/4}(G)$, $w_R(G - S_\rho) \geq n/24$ with probability at least $\frac{1}{2}$. □

6.3. Size lower bounds for Res(k). To prove the size lower bounds for Res(k) refutations of PHP_n^{cn} we first prove size lower bounds for the weak pigeonhole principle restricted to a robust graph, and then we reduce these principles to PHP_n^{cn}.

LEMMA 6.15. *For any $c > 1$ and $d > 0$, there exists $\epsilon > 0$ so that for all n sufficiently large, if $k \leq \sqrt{\log n / \log\log n}$ and G is a $(3/4, n/24)$-robust bipartite graph with vertex sets of sizes cn and n and $\Delta(G) \leq d \log n$, then $S_k(PHP(G)) \geq 2^{n^\epsilon}$.*

Proof. By Lemma 6.7, for each $i \in [k]$ and every i-DNF F,

$$\Pr_{\rho \in \mathcal{M}_{3/4}(G)} [F \restriction_\rho \neq 1] \leq 2^{-\frac{(\log e)(1-3/4)^i c(F)}{i(d \log n)^{i+1}}} = 2^{-\frac{(\log e) c(F)}{i \cdot 4^i (d \log n)^{i+1}}}.$$

In the interest of obtaining a better bound, we will not appeal to Corollary 3.4, but directly apply Theorem 3.3. We define sequences $s_0, \ldots, s_k$ and $p_1, \ldots, p_k$ for use in the switching lemma. Set $s_0 = \frac{3}{4k}(n/24 - 1)$. For each $i \in [k]$, set

$$s_i = \left(\frac{\log e}{2i4^i (d \log n)^{i+1}} \right) s_{i-1}.$$

For each $i \in [k]$ set $p_i = 2^{-2s_i}$. For any i-DNF F so that $c(F) > s_{i-1}$, we have the following inequality:

$$\Pr_{\rho \in \mathcal{M}_{3/4}(G)} [F \restriction_\rho \neq 1] < 2^{-\frac{(\log e) s_{i-1}}{i \cdot 4^i (d \log n)^{i+1}}} = 2^{-2\frac{(\log e) s_{i-1}}{2i4^i (d \log n)^{i+1}}} = 2^{-2s_i} = p_i.$$

It can be shown that there exists $\epsilon > 0$ so that for sufficiently large n, $s_k \geq n^\epsilon$. To avoid distraction, we show this in Lemma 6.17 at the end of this subsection. Suppose that Γ is a Res(k) refutation of $PHP(G)$ of size less than 2^{n^ϵ}.

By an application of Theorem 3.3 and the union bound, we have

$$\Pr_{\rho\in\mathcal{M}_{3/4}(G)}\left[\exists F\in\Gamma,\ h(F\restriction_\rho)>\sum_{i=0}^{k-1}s_i\right]\le 2^{n^\epsilon}\sum_{i=1}^{k}p_i2^{\sum_{j=i}^{k-1}s_j}$$

$$\le 2^{s_k}\sum_{i=1}^{k}p_i2^{\sum_{j=i}^{k-1}s_j}=\sum_{i=1}^{k}p_i2^{\sum_{j=i}^{k}s_j}.$$

We now bound $p_i2^{\sum_{j=i}^{k}s_j}$ for each $i>0$. For each i, $s_{i+1}<\frac{1}{4}s_i$ so $\sum_{j=i}^{k-1}s_j\le\frac{4}{3}s_i$. This gives us the following inequality:

$$p_i2^{\sum_{j=i}^{k-1}s_j}=2^{\sum_{j=i}^{k-1}s_j-2s_i}\le 2^{(4/3-2)s_i}=2^{-(2/3)s_i}\le 2^{-(2/3)s_k}\le 2^{-(2/3)n^\epsilon}.$$

Therefore,

$$\begin{aligned}&\Pr_{\rho\in\mathcal{M}_{3/4}(G)}\left[\exists F\in\Gamma,\ h(F\restriction_\rho)>(n/24-1)/k\right]\\&\le\Pr_{\rho\in\mathcal{M}_{3/4}(G)}\left[\exists F\in\Gamma,\ h(F\restriction_\rho)>\sum_{i=0}^{k-1}s_i\right]\\&\le\sum_{i=1}^{k}p_i2^{\sum_{j=i}^{k-1}s_j}\le\sum_{i=1}^{k}2^{-(2/3)n^\epsilon}\le k2^{-(2/3)n^\epsilon}=2^{\log k-(2/3)n^\epsilon}.\end{aligned}$$

For n sufficiently large, this probability is strictly less than $\frac{1}{2}$. Because G is $(3/4,n/24)$-robust, for $\rho\in\mathcal{M}_{3/4}(G)$, with probability at least $\frac{1}{2}$, $w_R(PHP(G)\restriction_\rho)\ge n/24$. Thus, there is a ρ so that $w_R(PHP(G)\restriction_\rho)\ge n/24$ and for all $F\in\Gamma$, $h(F\restriction_\rho)\le\frac{1}{k}(n/24-1)$. This is a contradiction, because by Corollary 5.2, there is a resolution refutation of $PHP(G)\restriction_\rho$ of width $\le n/24-1$. □

THEOREM 6.16. *For each $c>1$, there exists $\epsilon>0$ so that for all n sufficiently large, if $k\le\sqrt{\log n/\log\log n}$, then every Res($k$) refutation of PHP_n^{cn} has size at least 2^{n^ϵ}.*

Proof. Apply Lemma 6.9 and choose d so that for sufficiently large n, there exists a $(3/4,n/24)$-robust graph G on vertex sets cn and n, with $\Delta(G)\le d\log n$. By Lemma 6.15, there exists $\epsilon>0$ so that for $k\le\sqrt{\log n/\log\log n}$, $S_k(PHP(G))\ge 2^{n^\epsilon}$. Because $PHP(G)$ can be obtained by setting some of the variables of PHP_n^{cn} to 0, every Res(k) refutation of PHP_n^{cn} can be converted into a Res(k) refutation of $PHP(G)$ of the same or lesser size. Therefore, all Res(k) refutations of PHP_n^{cn} must have size at least 2^{n^ϵ}. □

Now we prove the lower bound on the number s_k that we used in Lemma 6.15. The constants are *not* optimized.

LEMMA 6.17. *There exists $\epsilon>0$, so that for all n sufficiently large, with $k\le\sqrt{\log n/\log\log n}$ and $s_0,\dots,s_k$ defined as in the proof of Lemma* 6.15, *$s_k\ge n^\epsilon$.*

Proof. Unwinding the recursive definition of the s_i's gives the following equality:

$$s_k=\frac{1}{2^k}(\log e)^k\frac{1}{k!}\left(\frac{1}{4}\right)^{\sum_{j=1}^{k}j}\left(\frac{1}{d\log n}\right)^{\sum_{j=2}^{k+1}j}\frac{3}{4k}(n/24-1).$$

Because $k\le\sqrt{\log n/\log\log n}$, we have that $\frac{1}{2^k}(\log e)^k\frac{1}{k!}\left(\frac{1}{4}\right)^{\sum_{j=1}^{k}j}\frac{3}{4k}=n^{-o(1)}$.

$$\begin{aligned}s_k&=n^{-o(1)}(1/d\log n)^{(k+2)(k+1)/2}(n/24-1)\\&=n^{-o(1)}2^{-(\log(d\log n))(k^2+3k+2)/2}(n/24-1).\end{aligned}$$

Because $k \leq \sqrt{\log n / \log\log n}$ and d is a constant, thus for n sufficiently large, $(\log(d \log n))(k^2 + 3k + 2)/2 = (\log n)(1 + o(1))/2$. Therefore,

$$s_k = n^{-o(1)} 2^{-(\log n)(1+o(1))/2}(n/24 - 1),$$

and there exists $\epsilon > 0$ so that for all n sufficiently large, $s_k \geq n^\epsilon$. □

7. Lower bounds for random CNFs. It is well known that, in some cases, randomly generated sets of clauses require exponentially large resolution refutations; see [13, 5, 8]. We extend these results by giving exponential lower bounds for the size of Res(k) refutations of randomly chosen sets of width $4k^2 + 2$ clauses.

DEFINITION 7.1. *Let n, Δ, and w be given. The distribution $\mathcal{F}_w^{n,\Delta}$ is defined by choosing $\Delta \cdot n$ many clauses independently, with repetitions, from the set of all $\binom{n}{w}2^w$ clauses of width w.*

Our main result for this section is the following.

THEOREM 7.2. *For any $\epsilon \in [0, \frac{1}{6})$, there exists $\delta > 0$, so that for n sufficiently large and for $\Delta = n^\epsilon$,*

$$\Pr_{F \in \mathcal{F}_{4k^2+2}^{n,\Delta}} \left[S_k(F) \leq 2^{n^\delta} \right] = o(1).$$

The reason that our proof does not give lower bounds for refutations of random 3-CNFs in Res(k) is that, on one hand, we want our random restrictions to have a good chance of satisfying a fixed k-term (so we can apply the switching lemma), but on the other hand, the restrictions should have little probability of falsifying any of the initial clauses (this would make the restricted set of clauses trivial to refute). Because satisfying a k-term is equivalent to falsifying a k-clause, we can only work with initial clauses of width larger than k.

A set of clauses that, with constant probability, requires high width to refute after random restriction is called *robust*. Recall the distribution $\mathcal{D}_p$ from Definition 3.5.

DEFINITION 7.3. *Let F be a CNF in variables $x_1, \ldots, x_n$. We say that F is (p, r)* robust *if $\Pr_{\rho \in \mathcal{D}_p}[w_R(F \restriction \rho) \geq r] \geq 1/2$.*

It turns out that for sufficiently large w, a random w-CNF is almost surely robust. We state the result below and prove it in the following subsection.

LEMMA 7.4. *There exists a constant c so that for any constants w and t, $w \geq 2t + 2$, for every n sufficiently large, and for every $\epsilon \in [0, 1/2]$, if we set $\Delta = n^\epsilon$, then the following inequality holds:*

$$\Pr_{F \in \mathcal{F}_w^{n,\Delta}} \left[F \text{ is not } \left(n^{-1/t}, cn^{\frac{1-2\epsilon}{1+2\epsilon}} \right)\text{-robust} \right] = o(1).$$

We now prove the size lower bound. We set bits with probability $n^{-1/2k^2}$ so we can collapse k-DNFs but still have that most $4k^2 + 2$ CNFs are robust. For each $k \geq 1$, let γ_k be the constant of corollary 3.7.

LEMMA 7.5. *Let n, r, w, and k be given. For sufficiently large n, if F is an $(n^{-1/2k^2}, r)$-robust w-CNF, then $S_k(F) \geq \frac{1}{4k} 2^{(\gamma_k (r-1)/k\sqrt{n})}$.*

Proof. Suppose that Γ is a Res(k) refutation of F of size at most $\frac{1}{4k} 2^{(\gamma_k (r-1)/k\sqrt{n})}$. By Corollary 3.7, with $p = n^{-1/2k^2}$ and $w = (r-1)/k$, we have that for every line F of Γ, $\Pr_{\rho \in \mathcal{D}_p}[h(F \restriction \rho) > (r-1)/k] \leq k 2^{-\gamma_k (r-1)/k\sqrt{n}}$. By the union bound we have that

$$\Pr_{\rho \in \mathcal{D}_p}[\exists F \in \Gamma \; h(F \restriction \rho) > (r-1)/k] \leq |\Gamma| \cdot k \cdot 2^{-\gamma_k (r-1)/k\sqrt{n}}$$

$$\leq \frac{1}{4k} 2^{(\gamma_k(r-1)/k\sqrt{n})} \cdot k \cdot 2^{-\gamma_k(r-1)/k\sqrt{n}} = \frac{1}{4}.$$

Because F is (p,r)-robust, with probability at least $\frac{1}{2}$ over choices of ρ, then $w_R(F \restriction_\rho) \geq r$. Therefore, we may choose $\rho \in \mathcal{D}_p$ so that $w_R(F \restriction_\rho) \geq r$ and for all $F \in \Gamma$, $h(F \restriction_\rho) \leq (r-1)/k$. This is a contradiction because by Corollary 5.2 there should be a width $r-1$ resolution refutation of $F \restriction_\rho$. □

Combining Lemmas 7.4 and 7.5 with $t = 2k^2$, $w = 4k^2 + 2$, and $r = cn^{\frac{1-2\epsilon}{1+2\epsilon}}$ shows that a random $(4k^2+2)$-CNF almost surely requires exponential size to refute in Res(k).

COROLLARY 7.6. *There exists a constant c so that for every k, for every n sufficiently large, and for $\epsilon \in [0, 1/2]$, if we set $\Delta = n^\epsilon$, then the following inequality holds:*

$$\Pr_{F \in \mathcal{F}^{n,\Delta}_{4k^2+2}} \left[S_k(F) \leq 2^{\gamma_k(cn^{\frac{1-2\epsilon}{1+2\epsilon}}-1)/k\sqrt{n}} \right] = o(1).$$

This gives an exponential lower bound only when $\frac{1-2\epsilon}{1+2\epsilon} > \frac{1}{2}$. This holds exactly for $\epsilon \in [0, \frac{1}{6})$.

THEOREM 7.7. *For any $\epsilon \in [0, \frac{1}{6})$, there exists $\delta > 0$, so that for n sufficiently large and for $\Delta = n^\epsilon$,*

$$\Pr_{F \in \mathcal{F}^{n,\Delta}_{4k^2+2}} \left[S_k(F) \leq 2^{n^\delta} \right] = o(1).$$

7.1. Robustness of random CNFs. In this section we show that for appropriate clause densities, a random w-CNF is almost surely robust.

We begin with a width bound for resolution refutations of random 3-CNFs given by Ben-Sasson and Wigderson.

THEOREM 7.8 (see [8]). *There exists a constant c, so that for all n, and for all $\epsilon \in [0, 1/2]$ with $\Delta = n^\epsilon$, the following inequality holds:*

$$\Pr_{F \in \mathcal{F}^{n,\Delta}_3} \left[w_R(F) \leq cn^{\frac{1-2\epsilon}{1+2\epsilon}} \right] = o(1).$$

LEMMA 7.9. *There exists a constant c so that for any constants w and t with $w \geq 2t+2$, for every n sufficiently large, and for $\epsilon \in [0, 1/2]$, if we set $\Delta = n^\epsilon$ and $p = n^{-1/t}$, then the following inequality holds:*

$$\Pr_{\substack{F \in \mathcal{F}^{n,\Delta}_w \\ \rho \in \mathcal{D}_p}} \left[w_R(F \restriction_\rho) \leq cn^{\frac{1-2\epsilon}{1+2\epsilon}} \right] = o(1).$$

Proof. Let w, t, n, ϵ be given as above and set $\Delta = n^\epsilon$ and $p = n^{-1/t}$.

Because the expected size of $\mathrm{dom}(\rho)$ is pn, the Chernoff bounds show that the size of $\mathrm{dom}(\rho)$ exceeds $2pn = 2n^{1-\frac{1}{t}}$ with probability at most $e^{-n^{1-1/t}/4} = o(1)$.

Let C be a fixed clause of width w that contains no opposite literals. When we choose $\rho \in \mathcal{D}_p$, the probability that the domain of ρ contains at least $w-2$ variables of C is at most $\binom{w}{2} p^{w-2}$. Because w is a constant, this probability is $O\left(n^{-(w-2)/t}\right)$. Because $w \geq 2t+2$, this probability is $O\left(n^{-2}\right)$. For any fixed w-CNF F on Δn many clauses, an application of the union bound shows that there is some clause with

$\geq w-2$ of its variables in the restriction with probability $O(\Delta n \cdot n^{-2}) = o(1)$. Because this calculation holds for every w-CNF of Δn many clauses, we have that

$$\Pr_{\substack{F \in \mathcal{F}_w^{n,\Delta} \\ \rho \in \mathcal{D}_p}} [\exists C \in F,\ |vars(C) \setminus \mathrm{dom}(\rho)| \leq 2] = o(1).$$

Fix a restriction ρ so that $\mathrm{dom}(\rho) \leq 2n^{1-\frac{1}{t}}$ and let $n' = n - |\mathrm{dom}(\rho)|$. Conditioned on the event that for all $i \in [\Delta]$, $|vars(C_i) \setminus \mathrm{dom}(\rho)| \geq 3$, $F \restriction_\rho$ is subsumed by a random 3-CNF distributed as $\mathcal{F}_3^{n',\Delta}$. (To see this, consider the distribution on 3-CNFs that chooses three literals unset by ρ from each C_i.) Choose ϵ' so that $n^\epsilon = (n')^{\epsilon'}$. Adding up the conditional probabilities and applying Theorem 7.8 shows that

$$\Pr_{\substack{F \in \mathcal{F}_w^{n,\Delta} \\ \rho \in \mathcal{D}_p}} \left[w_R(F \restriction_\rho) \leq c {n'}^{\frac{1-2\epsilon'}{1+2\epsilon'}} \right] = o(1).$$

Because $n' \geq n - 2n^{1-1/t}$, we may choose c' so that

$$\Pr_{\substack{F \in \mathcal{F}_w^{n,\Delta} \\ \rho \in \mathcal{D}_p}} \left[w_R(F \restriction_\rho) \leq c' n^{\frac{1-2\epsilon}{1+2\epsilon}} \right] = o(1). \qquad \square$$

From Lemma 7.9, an averaging argument yields the following phrasing of Lemma 7.4.

LEMMA 7.10. *There exists a constant c so that for any constants w and t, $w \geq 2t+2$, for every n sufficiently large, and for $\epsilon \in [0, 1/2]$, if we set $\Delta = n^\epsilon$ and let $p = n^{-1/t}$, then the following inequality holds:*

$$\Pr_{F \in \mathcal{F}_w^{n,\Delta}} \left[\Pr_{\rho \in \mathcal{D}_p} \left[w_R(F \restriction_\rho) \leq c n^{\frac{1-2\epsilon}{1+2\epsilon}} \right] \geq 1/2 \right] = o(1).$$

8. Separation between Res(k) and Res($k+1$). In this section we show that for each constant k, there is an $\epsilon_k > 0$ and a family of unsatisfiable CNFs which have polynomial-size Res$(k+1)$ refutations but which require size $2^{n^{\epsilon_k}}$ to refute in Res(k). The unsatisfiable clauses are a variation of the graph ordering tautologies [19, 9].

DEFINITION 8.1. *Let G be an undirected graph. For each vertex u of G, let $N(u)$ denote the set of neighbors of u in G. For each ordered pair of vertices $(u, v) \in V(G)^2$, with $u \neq v$, let there be a propositional variable $X_{u,v}$.*

The graph ordering principle for G, $GOP(G)$, *is the following set of clauses:*

(1) *The relation X is transitive: for all $u, v, w \in V(G)$, $X_{u,v} \wedge X_{v,w} \to X_{u,w}$.*
(2) *The relation X is antisymmetric: for all $u, v \in V(G)$ with $u \neq v$, $\neg X_{u,v} \vee \neg X_{v,u}$.*
(3) *There is no locally X-minimal element: for every $u \in V(G)$, $\bigvee_{v \in N(u)} X_{v,u}$.*

The k-fold graph ordering principle of G, $GOP^k(G)$, *is obtained by replacing each variable $X_{u,v}$ by a conjunction of k variables, $X^1_{u,v}, \ldots, X^k_{u,v}$, and then using the distributive rule and DeMorgan's law to express this as a set of clauses.*

Notice that for a graph G on n vertices with maximum degree d, the principle $GOP(G)$ consists of $O(n^3)$ many clauses each of width at most $\max\{3, d\}$. Therefore, for any graph G on n vertices with maximum degree d, the principle $GOP^k(G)$ has size $O(n^3 k^d)$.

It is readily shown that, for any graph G, the principle $GOP(G)$ has polynomial-size resolution refutations. Furthermore, these refutations can be transformed into Res$(k+1)$ refutations of $GOP^{k+1}(G)$, as shown in Lemma 8.4. On the other hand,

we will also prove that Res(k) refutations of $GOP^{k+1}(G)$ require exponential size for certain graphs.

THEOREM 8.2. *Let k be a positive integer. There exist constants $c > 0$, $\epsilon_k > 0$ and a family of graphs G on n vertices (for n sufficiently large) with maximum degree $c \log n$ so that Res(k) refutations of $GOP^{k+1}(G)$ require size at least $2^{\Omega(n^{\epsilon_k})}$.*

8.1. The upper bounds. We build Res(k) refutations for $GOP^k(G)$ from resolution refutations of $GOP(G)$.

The resolution refutation of $GOP(G)$ is a slight variation of the resolution refutation of GT_n [19, 9].

LEMMA 8.3. *Let G be an n vertex graph. There is a resolution refutation of $GOP(G)$ of size $O(n^3)$.*

Proof. To construct the resolution refutation of $GOP(G)$, we iteratively derive the formulas $\bigvee_{\substack{i\in[l,n]\\ i\neq j}} X_{i,j}$ for all i, j with $1 \leq l \leq j \leq n$. The clauses $\bigvee_{\substack{i\in[n]\\ i\neq j}} X_{i,j}$ are derived by weakening the hypotheses. We proceed in stages as l ranges from 1 up to n. At stage l, for $j = l$, we have $\bigvee_{i\in[l+l,n]} X_{i,l}$. For $j \neq l$, we resolve $\bigvee_{i\in[l+1,n]} X_{i,l}$ with the transitivity axioms $\neg X_{i,l} \vee \neg X_{l,j} \vee X_{i,j}$ to obtain $\neg X_{l,j} \vee X_{j,l} \bigvee_{\substack{i\in[l+1,n]\\ i\neq j}} X_{i,j}$. This clause is resolved with $\neg X_{l,j} \vee \neg X_{j,l}$ and $\bigvee_{\substack{i\in[l,n]\\ i\neq j}} X_{i,j}$ to obtain $\bigvee_{i=l+1}^{n} X_{i,j}$. At stage n, with $j = n$, we have derived the empty clause. This refutation clearly has size $O(n^3)$. □

LEMMA 8.4. *For each k, and for every G with n vertices and degree at most $d \geq 3$, $GOP^k(G)$ has a Res(k) refutation of size $O(n^3 k^d)$.*

Proof. Let τ be the operation that replaces $X_{u,v}$ by $\bigwedge_{i=1}^{k} X^i_{u,v}$ and $\neg X_{u,v}$ by $\bigvee_{i=1}^{k} \neg X^i_{u,v}$.

Let Γ be the size $O(n^3)$ resolution refutation of $GOP(G)$ given above, and remove all of its weakening inferences. If we apply the transformation τ to the refutation, we obtain a Res(k) refutation of $\tau(GOP(G))$.

From the clauses of $GOP^k(G)$ we can derive the k-DNFs of $\tau(GOP(G))$ by a sequence of $O(k^d)$ many AND-introduction inferences per formula. Thus, we have a Res(k) refutation of $GOP^k(G)$ of the claimed size. □

8.2. Random restrictions. In this subsection we define a distribution on partial assignments so that i-DNFs with high cover number are satisfied with high probability. The idea is to randomly color the graph with $4k$ many colors, and then, between vertices u and v of distinct color classes, uniformly choose an assignment to $X^1_{u,v}, \ldots, X^{k+1}_{u,v}, X^1_{v,u}, \ldots, X^{k+1}_{v,u}$ which makes both $\bigwedge_{i=1}^{k+1} X^i_{u,v}$ and $\bigwedge_{i=1}^{k+1} X^i_{v,u}$ false.

DEFINITION 8.5. *Let $k \geq 1$ be given. Let G be a graph. The distribution $\mathcal{P}_{k+1}(G)$ on partial assignments ρ to the variables of $GOP^{k+1}(G)$ is given by the following experiment.*

For each $(u, v) \in V(G)^2$, let $\sigma^{u,v}_\rho$ be chosen uniformly among $0, 1$ assignments to $X^1_{u,v}, \ldots, X^{k+1}_{u,v}$ so that for at least one $i \in [k + 1]$, $\sigma^{u,v}_\rho(X^i_{u,v}) = 0$.

Select a random coloring of $V(G)$ by $4k$ many colors, $c_\rho : V(G) \to [4k]$.

The partial assignment, ρ, is defined as

$$\rho = \bigcup_{\substack{(u,v)\in V(G)^2\\ c_\rho(u)\neq c_\rho(v)}} \sigma^{u,v}_\rho.$$

The auxiliary total assignment, σ_ρ, is defined as

$$\sigma_\rho = \bigcup_{(u,v)\in V(G)^2} \sigma_\rho^{u,v}.$$

If we let B_ρ be the set of edges which are bichromatic under the coloring c_ρ, then $GOP^{k+1}(G) \restriction_\rho$ is $GOP^{k+1}(G \setminus B_\rho)$. Moreover, we have the following lemma, which the reader can easily check.

LEMMA 8.6. *Let G be a graph. Let $\rho \in \mathcal{P}_{k+1}(G)$ be given. Let B_ρ be the set of edges of G that are bichromatic under c_ρ. Let $G_1, \ldots, G_m$ be the connected components of $G \setminus B_\rho$.*

$$GOP^{k+1}(G) \restriction_\rho = \bigcup_{j=1}^{m} GOP^{k+1}(G_j).$$

Formulas with high cover number contain many variable-disjoint terms, but the events of satisfying these terms with $\rho \in \mathcal{P}_{k+1}(G)$ are not necessarily independent. To obtain independence, we look at the pairs of vertices involved with the literals of the terms. Remember that in the definition of $GOP(G)$, there are no variables $X_{u,u}$.

DEFINITION 8.7. *Let $X^i_{u,v}$ be a variable of $GOP^{k+1}(G)$. The* underlying pair *of $X^i_{u,v}$ is the set $\{u,v\}$. The* underlying ordered pair *of $X^i_{u,v}$ is (u,v). Let T be a term. The set of* vertex pairs of T, P_T, *is defined as*

$$P_T = \{\{u,v\} \mid \ \{u,v\} \text{ is the underlying pair of a variable in } T\}.$$

The set of vertices of T, S_T, *is defined as $S_T = \bigcup P_T$.*

We use combinatorial sunflowers to obtain independence between the events of satisfying terms of an i-DNF with high cover number. To guarantee that such a system exists, we apply the Erdös–Rado lemma.

DEFINITION 8.8. *A (p,l)* sunflower *is a collection of sets $P_1, \ldots, P_p$, each of size $\leq l$, so that there exists a set C so that $P_i \cap P_j = C$ for all $i,j \in [p]$, $i \neq j$. The set C is called the* core *of the sunflower.*

THEOREM 8.9 (see [16, 22]). *Let l be given. Let $\mathcal{Z}$ be a family of M distinct sets, each with cardinality $\leq l$. $\mathcal{Z}$ contains a (p,l) sunflower where $p \geq \left(\frac{M}{l!}\right)^{\frac{1}{l}}$.*

DEFINITION 8.10. *Let $T_1, \ldots, T_t$ be terms in the variables of $GOP^{k+1}(G)$. We say that the terms are* sufficiently independent *if the following conditions hold:*

1. *For $i,j \in [t]$, if $i \neq j$, then $S_{T_i} \neq S_{T_j}$.*
2. *The family $\{S_{T_i} \mid 1 \leq i \leq t\}$ forms a sunflower with core C.*
3. *For each $i \in [t]$, each $\{u,v\} \in P_{T_i}$, $\{u,v\} \not\subseteq C$.*

LEMMA 8.11. *Let $T_1, \ldots, T_t$ be a sufficiently independent set of terms. The sets P_{T_i}, $1 \leq i \leq t$, are disjoint.*

Proof. Let i,j, $1 \leq i < j \leq t$, be given and let C denote the core of the sunflower. Suppose that $\{u,v\} \in P_{T_i} \cap P_{T_j}$. We then have that $\{u,v\} \subseteq S_{T_i} \cap S_{T_j}$, so $\{u,v\} \subseteq C$. Therefore, by the third property of sufficient independence, $\{u,v\} \notin P_{T_i}$—a contradiction. □

We begin the task of showing that a DNF with high cover number is likely to be satisfied by a random restriction. The quality of our bounds is most affected by the use of the sunflower lemma, and the particular constants we obtain at other points have limited impact. Therefore, to conserve space and readability, we will not optimize many of the probabilities involved.

LEMMA 8.12. *Let k be given. There exist constants $\beta_k > 0$ and $c_k > 0$ so that for every k-DNF F in the variables of $GOP^{k+1}(G)$, F contains a sufficiently independent set of size at least $\beta_k(c(F))^{\frac{1}{2k}} - c_k$.*

Proof. F contains a set of $s = c(F)/k$ many variable-disjoint terms, $T_1, \ldots, T_s$. It is possible that $S_{T_m} = S_{T_l}$ for some $m \neq l$. However, because all terms have size at most k, for each i, $|S_{T_i}| \leq 2k$, and a set of $\leq 2k$ many vertices can be the underlying set of fewer than $\big((k+1)4k^2\big)^k$ many different variable-disjoint terms (since a variable $X^i_{u,v}$ is determined by an ordered pair $(u,v) \in [S_T]^2$ and $i \in [k+1]$). Therefore, there is a subcollection of $\frac{s}{((k+1)4k^2)^k}$ many variable-disjoint terms whose underlying sets of vertices are distinct.

Because the underlying sets of vertices have size at most $2k$, we can apply the sunflower lemma to find $s' = \big(\frac{s}{((k+1)4k^2)^k(2k)!}\big)^{\frac{1}{2k}} = \big(\frac{c(F)}{k((k+1)4k^2)^k(2k)!}\big)^{\frac{1}{2k}}$ many terms whose sets of underlying vertices form an $(s', 2k)$ sunflower. We rename these terms $T_1, \ldots, T_{s'}$.

Let C be the core of the sunflower $S_{T_1}, \ldots, S_{T_{s'}}$. Notice that $|C| \leq 2k$. Call a variable *bad* if both of its underlying vertices belong to C. There are fewer than $2k^2$ many unordered pairs of vertices contained in C, and each is the underlying pair of exactly $2(k+1)$ many variables. Therefore, there are fewer than $2(k+1)\cdot 2k^2 = 4k^2(k+1)$ many variables whose underlying vertices are both in C. The terms $T_1, \ldots, T_{s'}$ are variable-disjoint, so each bad variable appears in at most one term, and when we remove all terms containing a bad variable, we obtain a sufficiently independent set of terms of size $s' - 4k^2(k+1) = \big(\frac{c(F)}{k(2k)!((k+1)4k^2)^k}\big)^{\frac{1}{2k}} - 4k^2(k+1)$. ∎

Before we bound the probability of satisfying a DNF with high covering number, we make a few observations.

FACT 1. *Let T be a term, and let $\rho \in \mathcal{P}_{k+1}(G)$ be given. $T \restriction_\rho = 1$ if and only if the following two events occur:* (i) *$T \restriction_{\sigma_\rho} = 1$, and* (ii) *for each $\{u,v\} \in P_T$, $c_\rho(u) \neq c_\rho(v)$.*

For each term T in a Res(k) refutation of $GOP^{k+1}(G)$, because T contains at most k literals, there is a nonzero chance that it will be satisfied by σ_ρ when ρ is a random restriction chosen according to $\mathcal{P}_{k+1}(G)$. This is made precise in the following lemma. Recall that by construction, $\bigwedge_{i=1}^{k+1} X^i_{u,v} \restriction_{\sigma_\rho} = 0$, so the lemma fails for terms of size $k+1$ (as it should, since we are separating Res($k+1$) from Res(k)). The argument is similar to that in Lemma 4.4.

LEMMA 8.13. *Let T be a term of size at most k.*

$$\Pr_{\rho \in \mathcal{P}_{k+1}(G)}\left[T \restriction_{\sigma_\rho} = 1\right] \geq \frac{1}{3^k}.$$

Proof. Order the literals of T as $l_1, \ldots, l_k$. For each j, $1 \leq j \leq k$, if we condition on the event that each $l_1, \ldots, l_{j-1}$ is satisfied, then the probability of l_j being satisfied is at least $\frac{1}{3}$. This is because in the worst case, l_j is a literal $X^{i_j}_{u,v}$ and the other literals are $X^{i_1}_{u,v}, \ldots, X^{i_{j-1}}_{u,v}$, and in this case, the probability that $X^{i_j}_{u,v}$ is satisfied by σ_ρ is at least $\frac{1}{3}$. ∎

LEMMA 8.14. *Let G be a graph and let k be a positive integer. Let F be a k-DNF which contains t sufficiently independent terms.*

$$\Pr_{\rho \in \mathcal{P}_{k+1}(G)}\left[F \restriction_\rho \neq 1\right] \leq \left(1 - \frac{1}{3^k 2^{2k}}\right)^t.$$

Proof. Let $T_1, \ldots, T_t$ be the sufficiently independent terms of F. Let C be the core

of the sunflower $S_{T_1}, \ldots, S_{T_t}$. Fix a coloring of the vertices in the core, $\chi : C \to [4k]$. Condition on the event that $c_\rho \restriction_C = \chi$.

We now lower-bound the probability that a given term T of the sufficiently independent set is satisfied. First, we bound the probability that every underlying edge of T is bichromatic. Note that by property (3) of sufficient independence, for all $\{u, v\} \in P_T$, $\{u, v\} \not\subseteq C$, so it suffices to bound the probability that the vertices in $S_T \setminus C$ receive distinct colors not in the range of χ. Therefore, the probability that every pair in P_T is bichromatic, conditioned on $c_\rho \restriction_C = \chi$, is at least $\frac{1}{2^{2k}}$. Because T contains at most k literals, the probability that $T \restriction_{\sigma_\rho} = 1$ is at least $\frac{1}{3^k}$. These two events are independent, so we have $\Pr_{\rho \in \mathcal{P}_{k+1}(G)}[T \restriction_\rho = 1 \mid c_\rho \restriction_C = \chi] \geq \frac{1}{2^{2k}} \frac{1}{3^k}$.

Now we show that (when we condition on the event that $c_\rho \restriction_C = \chi$) the events $T_i \restriction_\rho = 1$ are totally independent. Because the terms share no underlying pairs, the events $T_i \restriction_{\sigma_\rho}$ are independent of the satisfaction of other terms. The events "for each $\{u, v\} \in P_{T_i}$, $c_\rho(u) \neq c_\rho(v)$" are independent of the satisfaction of other terms. This is because once we condition on the event $c_\rho \restriction_C = \chi$, the probability that every pair of P_{T_i} is bichromatic under c_ρ depends only on the values that c_ρ takes on $S_{T_i} \setminus C$ and, for all $i \neq j$, $S_{T_i} \cap S_{T_j} = C$.

Combining the results of the previous two paragraphs shows that

$$\Pr_{\rho \in \mathcal{P}_{k+1}(G)}[F \restriction_\rho \neq 1 \mid c_\rho \restriction_C = \chi] \leq \left(1 - 1/3^k 2^{2k}\right)^t.$$

Because this holds for all colorings $\chi : C \to [4k]$, we have that

$$\Pr_{\rho \in \mathcal{P}_{k+1}(G)}[F \restriction_\rho \neq 1] \leq \left(1 - 1/3^k 2^{2k}\right)^t. \qquad \square$$

We now have the lemma relating cover number to the probability that a restriction satisfies a k-DNF.

LEMMA 8.15. *For each k there exist positive constants δ, γ, and d so that for any k-DNF F,*

$$\Pr_{\rho \in \mathcal{P}_{k+1}(G)}[F \restriction_\rho \neq 1] \leq d 2^{-\delta (c(F))^\gamma}.$$

Proof. By Lemma 8.12, F contains a sufficiently independent set of size at least $\beta_k (c(F))^{\frac{1}{2k}} - c_k$.

By Lemma 8.14,

$$\Pr_{\rho \in \mathcal{P}_{k+1}(G)}[F \restriction_\rho \neq 1] \leq \left(1 - \frac{1}{3^k 2^{2k}}\right)^{\beta_k (c(F))^{\frac{1}{2k}} - c_k}.$$

Because k is fixed, this concludes the proof with $\delta = -\beta_k \log\left(1 - \frac{1}{3^k 2^{2k}}\right)$, $\gamma = 1/2k$, and $d = \left(1 - \frac{1}{3^k 2^{2k}}\right)^{-c_k}$. $\square$

8.3. Width lower bound for resolution. In this subsection we show that for each n, if G is a graph on n vertices satisfying a certain expansion-like property, then $w_R(GOP^{k+1}(G)) = \Omega(n)$. Combining this with a probabilistic calculation will show that there exist graphs G so that for $\rho \in \mathcal{P}_{k+1}(G)$, with probability at least $\frac{1}{2}$, $w_R(GOP^{k+1}(G) \restriction_\rho) = \Omega(n)$.

The proof of the resolution width bound is similar to the one used by Bonet and Galesi for the GT_n principles [9]. They worked with complete graphs, but we do not because the principles $GOP^2(K_n)$ have sizes in excess of 2^n. Fortunately, for the

proof technique to work, G need not be complete but instead must have the following property.

DEFINITION 8.16. *Let G be an undirected graph on n-vertices. We say that G is ϵ-neighborly if, between every pair of disjoint sets of vertices, $A, B \subseteq V(G)$ with $|A|, |B| \geq \epsilon n$, there exists an edge joining A and B.*

We now show that resolution refutations of $GOP(G)$ require large width when G is a connected, neighborly graph.

LEMMA 8.17. *If G is a connected graph of n vertices that is ϵ-neighborly, then every resolution refutation of $GOP(G)$ contains a clause of width $\left(\frac{1-3\epsilon}{6}\right) n$.*

Proof. We begin by defining the "measure" of a clause. A *critical truth assignment*, or *cta*, is an assignment to the variables of $GOP(G)$ which forms a total order on $V(G)$. For each $v \in V(G)$, let $C_v := \bigvee_{u \in N(v)} X_{u,v}$, and for each $I \subseteq V(G)$, $C_I := \bigwedge_{v \in I} C_v$. Let C be a clause. The *measure of C, $\mu(C)$,* is the minimum cardinality of a set $I \subseteq V(G)$ so that for every cta α, if α satisfies C_I, then α satisfies C.

Notice that if a clause $A \vee B$ is the resolvent of $A \vee x$ and $B \vee \neg x$, then $\mu(A \vee B) \leq \mu(A \vee x) + \mu(A \vee \neg x)$. Because of this, we say that μ is *subadditive with respect to resolution.* If $A \subseteq B$, then we have that $\mu(B) \leq \mu(A)$, so μ is *decreasing with respect to subsumption.*

We now show that $\mu(\emptyset) = n$. Suppose otherwise, and let I be a subset of $V(G)$ with $|I| \leq n - 1$. Choose one vertex $v_0 \in V(G) \setminus I$ and let α be a total order which arises by taking a depth-first search of G starting with v_0. Clearly α satisfies C_I but α does not satisfy $\emptyset$.

Because every clause of $GOP(G)$ has measure either 0 or 1, the empty clause has measure n, and the measure is both subadditive with respect to resolution and decreasing with respect to subsumption, there must exist a clause C so that $\frac{n}{3} \leq \mu(C) \leq \frac{2n}{3}$. Suppose for the sake of contradiction that $w(C) < \frac{n-3\epsilon n}{6}$.

Let I be a minimal subset of $V(G)$ so that for every critical truth assignment α, if α satisfies C_I, then α satisfies C. Let $J = V(G) \setminus I$. Notice that $|I|, |J| \geq \frac{n}{3}$.

Let S be the set of vertices mentioned by variables of C. Clearly, $|S| \leq 2w(C) < 2\left(\frac{n-3\epsilon n}{6}\right) = \frac{n-3\epsilon n}{3}$. Therefore, $|I \setminus S| \geq \frac{n}{3} - \frac{n-3\epsilon n}{3} = \epsilon n$. Similarly, $|J \setminus S| \geq \epsilon n$. Because G is ϵ-neighborly, we may choose $u \in I \setminus S$ and $v \in J \setminus S$ so that $\{u, v\}$ is an edge of G.

Let α be a cta so that α satisfies $C_{I \setminus \{u\}}$, but α does not satisfy C_u and α does not satisfy C. Let β be the cta which arises by moving v to the front of the order given by α. For $w \in I$, $w \neq u$, β satisfies C_w because every predecessor of w in α is a predecessor of w in β. For u, β satisfies C_u because β satisfies $X_{v,u}$. However, β does not satisfy C because α does not satisfy C and no variable mentioning u or v appears in C. Therefore, β satisfies C_I but β does not satisfy C—a contradiction to the choice of I. □

A resolution refutation of $GOP^k(G)$, $k \geq 1$, can be transformed into a resolution refutation of $GOP(G)$ by setting the appropriate variables to 1. Applying a restriction does not increase the width of a resolution refutation, so we have the following corollary.

COROLLARY 8.18. *If G is a connected graph of n vertices that is ϵ-neighborly, then for all $k \geq 1$, $w_R(GOP^k(G)) \geq \left(\frac{1-3\epsilon}{6}\right) n$.*

8.4. Robust graphs.

DEFINITION 8.19. *We say that a graph G is r-robust if for ρ selected at random by $\mathcal{P}_{k+1}(G)$, with probability at least $\frac{3}{4}$, $w_R(GOP(G) \restriction_\rho) \geq r$.*

To guarantee that the restricted principle will require high width to refute, it suffices that the graph obtained by deleting the bichromatic edges should consist of large, neighborly connected components. Random graphs of degree $\Theta(\log n)$ have this property with high probability. This is shown in the following subsection.

LEMMA 8.20. *There exists a constant c so that for sufficiently large n, there exists an $\frac{n}{96k}$-robust graph G on n vertices with degree at most $c \log n$.*

The proof of this lemma is a standard probabilistic argument. The reader may skip its proof in section 8.4.1, and move directly to the proof of the lower bound in section 8.5.

8.4.1. Demonstration of robust graphs. An easy probabilistic argument shows that with very high probability, a random graph of expected degree $\Theta((1/\epsilon)\log n)$ is almost surely ϵ-neighborly.

LEMMA 8.21. *Let n and d be positive integers so that $d \leq n$ and let $p = d/n$. Let $G_{n,p}$ be the distribution on graphs on n vertices in which every edge is included with independent probability p. With probability $\leq e^{2\epsilon n(1+\ln(1/\epsilon))-d\epsilon^2 n}$, a graph selected according to $G_{n,p}$ is* not *ϵ-neighborly.*

Proof. There are fewer than $\binom{n}{\epsilon n}^2 \leq \left(\frac{en}{\epsilon n}\right)^{2\epsilon n} = e^{2\epsilon n(1+\ln(1/\epsilon))}$ many pairs of disjoint sets of ϵn many vertices. Each such pair has a chance of at most $(1-p)^{\epsilon^2 n^2}$ of being unconnected. However, $(1-p)^{\epsilon^2 n^2} = (1-\frac{d}{n})^{\epsilon^2 n^2} \leq e^{-d\epsilon^2 n}$, so by the union bound the probability is at most $e^{2\epsilon n(1+\ln(1/\epsilon))} e^{-d\epsilon^2 n} = e^{2\epsilon n(1+\ln(1/\epsilon))-d\epsilon^2 n}$. ∎

We now show that a random graph will probably have each component large and neighborly if we randomly partition it into vertex-induced subgraphs.

LEMMA 8.22. *For all $\epsilon > 0$ and all integers $k \geq 1$, there exists a constant c so that for sufficiently large n, there exists graph G with $\Delta(G) \leq 2c \log n$ so that upon the random partition of G into $4k$ many vertex-induced subgraphs, with probability at least $\frac{1}{2}$, each component has size at least $n/8k$ and is ϵ-neighborly.*

Proof. Let $p = \frac{c \log n}{n}$. We will solve for the value of c at the end. Consider the following experiment: Select a graph G according to the distribution $G_{n,p}$, and independently color each vertex with one of $4k$ colors. Then remove all bichromatic edges to form $4k$ vertex induced-subgraphs, $G_1, \ldots, G_{4k}$.

Let P be the probability that G has a vertex of degree $> 2c \log n$, or that one of the induced subgraphs has size $< \frac{n}{8k}$, is disconnected, or is not ϵ-neighborly. We now bound this probability.

Consider the probability that G has a vertex of degree $\geq 2c \log n$. By the Chernoff bounds, the probability of any one vertex having degree in excess of $2p(n-1)$ is no more than $e^{-p(n-1)/4} = e^{-c(\log n)(n-1)/4n}$. Therefore, the probability of there existing a vertex with degree in excess of $2p(n-1)$ is no more than $ne^{-c(\log n)(n-1)/4n}$.

The Chernoff bounds also allow us to bound the probability that any of the G_i's contain too few vertices. The probability that a given color class of the partition contains fewer than $\frac{1}{2} \cdot \frac{n}{4k} = \frac{n}{8k}$ vertices is bounded by $e^{-\frac{n}{64k}}$.

Once we condition upon all pieces of the partition containing at least $\frac{n}{8k}$ vertices, we can bound the probability that any induced subgraph is disconnected. Consider a fixed set of $s \geq \frac{n}{8k}$ many vertices, and condition upon the event that those vertices receive the same color in the partition. Each edge internal to the set is included with probability $\frac{c \log n}{n} = \frac{(cs/n)\log n}{s} \geq \frac{(c/8k)\log s}{s}$. By a standard result on the connectivity of random graphs (cf. [27]), each color class is disconnected with probability bounded by $O\left(1/n^{(c/8k)-1}\right)$.

Finally, we consider the probability that each of the components G_i is ϵ-neighborly. For a fixed set of $s \geq \frac{n}{8k}$ vertices, if we condition on the event that set forms a component after partition, each internal edge is included with probability $\frac{c \log n}{n} \geq \frac{(c/8k) \log s}{s}$. By Lemma 8.21, that means that the component is *not* ϵ-neighborly with probability at most $e^{2\epsilon s(1+\ln(1/\epsilon))-(c/8k)(\log s)\epsilon^2 s} = e^{-\Omega(n \log n)}$.

Therefore,

$$P \leq ne^{-c(\log n)(n-1)/4n} + 4ke^{-\frac{n}{64k}} + O\left(4k/n^{(c/8k)-1}\right) + e^{-\Omega(n \log n)}.$$

For a sufficiently large constant c, dependent only on k and ϵ, this is below $\frac{1}{4}$.

Therefore, by an averaging argument on the edge choices, there exists a graph G of maximum degree $\leq 2c \log n$ so that upon random partition of its vertices into $4k$ color classes, its induced subgraphs are each connected, of size $\geq \frac{n}{8k}$, and ϵ-neighborly with probability $\geq \frac{3}{4}$. ∎

We now prove Lemma 8.20.

Proof. Using Lemma 8.22, choose c so that for sufficiently large n, there exists graph G so that upon the random partition of G into $4k$ many vertex-induced subgraphs, with probability at least $\frac{3}{4}$, each component has size at least $n/8k$ and is $(\frac{1}{6})$-neighborly. Therefore we may choose $\rho \in \mathcal{P}_{k+1}(G)$ so that for each $i \in [4k]$, $w_R(GOP^{k+1}(G_i)) \geq \left(\frac{n}{8k}\right)\left(\frac{1-3\frac{1}{6}}{6}\right) = \frac{n}{96k}$ (by Corollary 8.18).

Let Γ be a resolution refutation of $GOP^{k+1}(G) \restriction_\rho$. By Lemma 8.6, $GOP^{k+1}(G) \restriction_\rho = \bigcup_{i=1}^{4k} GOP^{k+1}(G_i)$. However, for each $i, j \in [4k]$, $i \neq j$, we have that $GOP^{k+1}(G_i)$ and $GOP^{k+1}(G_j)$ are variable disjoint. Therefore, by Lemma 2.3, for some $i \in [4k]$, there exists a resolution refutation of $GOP^{k+1}(G_i)$ of width at most $w(\Gamma)$. However, by the preceding paragraph, each $GOP^{k+1}(G_i)$ requires width $\frac{n}{96k}$ to refute in resolution. Therefore $w(\Gamma) \geq \frac{n}{96k}$. ∎

8.5. The lower bound.

THEOREM 8.23. *Let k be given. There exist constants $c > 0$, $\epsilon_k > 0$ and a family of graphs G on n vertices (for n sufficiently large) with maximum degree $c \log n$ so that Res(k) refutations of $GOP^{k+1}(G)$ require size at least $2^{\Omega(n^{\epsilon_k})}$.*

Proof. Let k be given. Apply Lemma 8.20 and choose c so that for sufficiently large n, there exists a $\frac{n}{96k}$-robust graph G on n vertices with degree at most $c \log n$. By Lemma 8.15, there are positive constants d, δ, and γ so that for every k-DNF F $\Pr_{\rho \in \mathcal{P}_{k+1}(G)}[F \restriction_\rho \neq 1] \leq d2^{-\delta(c(F))^\gamma}$. By Corollary 3.4, with $s = (\frac{n}{96k} - 1)/k$, for every k-DNF F,

$$\Pr_{\rho \in \mathcal{P}_{k+1}(G)}[h(F \restriction_\rho) > (n/96k - 1)/k] \leq dk2^{-2\delta^k((n/96k-1)/k)^{\gamma^k}/4^k}.$$

Because d, γ, and δ depend only on k, there exists ϵ_k so that

$$\Pr_{\rho \in \mathcal{D}}[h(F \restriction_\rho) > (n/96k - 1)/k] \leq \frac{1}{2}2^{-n^{\epsilon_k}}.$$

Suppose for the sake of contradiction that Γ is a Res(k) refutation of $GOP^{K+1}(G)$ of size less than $2^{n^{\epsilon_k}}$. By the union bound, with probability at least $\frac{1}{2}$, every line F of Γ has $h(F \restriction_\rho) \leq (n/96k - 1)/k$. On the other hand, because G is $(n/96k)$-robust, $w_R(GOP^{k+1}(G) \restriction_\rho) \geq n/96k$ with probability at least $\frac{3}{4}$. So we may choose $\rho \in \mathcal{P}_{k+1}(G)$ so that $w_R(GOP^{k+1}(G) \restriction_\rho) \geq n/96k$, and for all lines F of Γ, $h(F \restriction_\rho) \leq (n/96k - 1)/k$. By Corollary 5.2, $GOP^{k+1}(G) \restriction_\rho$ has a resolution refutation of width at most $n/96k - 1$. This is a contradiction. ∎

9. Separating Res(k) and Res($k+1$) with constant width clauses. The separation between Res($k+1$) and Res(k) given by Theorem 8.23 uses sets of clauses whose maximum width is $\Theta(\log n)$. In this section we present a similar result which separates Res(k) and Res($k+1$) using constant width clauses.

DEFINITION 9.1. *Let $X_1, \ldots, X_k$ be propositional variables. The formula Odd $(X_1, \ldots, X_k)$ is the k-DNF expressing that the number of satisfied variables of $X_1, \ldots, X_k$ is odd. The formula Even $(X_1, \ldots, X_k)$ is the k-DNF expressing that the number of satisfied variables of $X_1, \ldots, X_k$ is even.*

The k-parity graph ordering principle of G, *$GOP^{\oplus k}(G)$, is obtained by replacing each literal $X_{u,v}$ by Odd $(X^1_{u,v}, \ldots, X^k_{u,v})$, replacing each literal $\neg X_{u,v}$ by Even $(X^1_{u,v}, \ldots, X^k_{u,v})$, and then using the distributive rule and DeMorgan's law to express this set of k-DNFs as a set of clauses.*

Because every clause of $GOP(G)$ contains at most $\max(d, 3)$ literals, every k-DNF in $GOP(G)[X_{u,v} \leftarrow \mathrm{Odd}(X^1_{u,v}, \ldots, X^k_{u,v}), \neg X_{u,v} \leftarrow \mathrm{Even}(X^1_{u,v}, \ldots, X^k_{u,v})]$ contains at most dk variables. When such a DNF is expressed as a set of clauses using the distributive rule, the set of clauses has size at most $2^{O(dk)}$ and each clause has width at most dk. Therefore, $GOP^{\oplus k}(G)$ contains at most $2^{O(dk)}n^3$ clauses, each of width at most dk.

For any graph G, the polynomial-size refutations of $GOP(G)$ can be transformed into Res($k+1$) refutations of $GOP^{\oplus(k+1)}(G)$. On the other hand, Res(k) refutations of $GOP^{\oplus(k+1)}(G)$ require exponential size for certain graphs.

THEOREM 9.2. *Let k be given. There exist constants $d > 0$, $\epsilon_k > 0$ and a family of graphs G on n vertices (for n sufficiently large) with maximum degree d so that Res(k) refutations of $GOP^{\oplus(k+1)}(G)$ require size at least $2^{\Omega(\epsilon_k n)}$.*

9.1. The upper bounds. We build Res(k) refutations for $GOP^{\oplus k}(G)$ from resolution refutations of $GOP(G)$.

DEFINITION 9.3. *Let k be a positive integer and let $X_1, \ldots, X_n$ be propositional variables. Let $X^1_1, \ldots, X^k_1, X^1_2, \ldots, X^k_n$ be new variables. Let σ be the mapping given by $\sigma(X_i) = Even(X^1_i, \ldots, X^k_i)$ and $\sigma(\neg X_i) = Odd(X^1_i, \ldots, X^k_i)$. For a clause $C = \bigvee_i l_i$, let $\sigma(C) = \bigvee_i \sigma(l_i)$.*

LEMMA 9.4. *Let k be a constant. There exists a constant c (dependent only on k) so that for all clauses $A \vee X_i$ and $B \vee \neg X_i$ in the variables $X_1, \ldots, X_n$, there is a derivation of $\sigma(A) \vee \sigma(B)$ from $\{\sigma(A) \vee \sigma(X_i), \sigma(B) \vee \sigma(\neg X_i)\}$ of size $\leq c$.*

Proof. By the completeness of Res(k), there is a Res(k) refutation of the pair of k-DNFs $\{\mathrm{Even}(X_1, \ldots, X_k), \mathrm{Odd}(X_1, \ldots, X_k)\}$. Let c be the minimum size of such a refutation. Because $\sigma(X_i) = \mathrm{Odd}(X^1_i, \ldots, X^k_i)$ and $\sigma(\neg X_i) = \mathrm{Even}(X^1_i, \ldots, X^k_i)$, there is a derivation of $\sigma(A) \vee \sigma(B)$ from $\{\sigma(A) \vee \sigma(X_i), \sigma(B) \vee \sigma(\neg X_i)\}$ of size greater than or equal to c. □

LEMMA 9.5. *For each k, there exists a constant c so that for every G with n vertices and degree at most $d \geq 3$, $GOP^{\oplus k}(G)$ has a Res(k) refutation of size $2^{O(dk)}n^3$.*

Proof. With the repeated application of AND-introduction inferences, $\sigma(GOP(G))$ can be derived from $GOP^{\oplus k}(G)$ in $2^{O(dk)}n^3$ many inferences. By Lemma 8.3, $GOP(G)$ has a refutation of size $O(n^3)$ so by Lemma 9.4, $\sigma(GOP(G))$ has a Res(k) refutation of size $O(cn^3)$. Therefore, $GOP^{\oplus k}(G)$ has a refutation of size $2^{O(dk)}n^3$. □

9.2. Random restrictions.

DEFINITION 9.6. *Let $k \geq 1$ be given, and let G be a graph. The distribution $\mathcal{P}_{k+1}(G)$ on partial assignments ρ to the variables of $GOP^{\oplus(k+1)}(G)$ is given by the*

following experiment:

For each $(u,v) \in V(G)^2$, choose $i \in \{1,\ldots,k+1\}$ uniformly and independently. For each $j \in \{1,\ldots,k\}$, $j \neq i$, set $X^j_{u,v}$ to 0 or 1, uniformly and independently.

LEMMA 9.7. *Let k be given; let F be a k-DNF in the variables of $GOP^{\oplus(k+1)}(G)$. There exist constants $\delta > 0$, dependent only on k, so that $\Pr_{\rho\in\mathcal{P}_{k+1}(G)}[F \restriction_\rho \neq 1] \leq 2^{-\delta\cdot c(F)}$*

Proof. We will say that two terms T and T' are *underlying-variable-disjoint* if whenever $X^i_{u,v} \in T$ and $X^{i'}_{u',v'} \in T'$, we have that $(u,v) \neq (u',v')$. Because F is a k-DNF, it contains at least $c(F)/k(k+1)$ many underlying-variable-disjoint terms. Each of these terms is satisfied with independent probability at least $1/4^k$ (consider setting each variable of a term in turn, the probability that a variable is set to 0 or 1 is always greater than or equal to $1/(k+1-(k-1)) = 1/2$). Therefore, $\Pr_{\rho\in\mathcal{P}_{k+1}(G)}[F \restriction_\rho \neq 1] \leq (1-1/4^k)^{c(F)/k(k+1)}$. □

When we apply a random restriction from $\mathcal{P}_{k+1}(G)$ to $GOP^{\oplus(k+1)}(G)$, we do not necessarily obtain an instance of $GOP(G)$. It is possible that some of the edge variables will become inverted. However, inverting some variables does not affect the width required for a resolution refutation, and we may apply Lemma 8.17.

COROLLARY 9.8. *If G is a connected graph that is ϵ-neighborly, then for all $k \geq 1$ and for all $\rho \in \mathcal{P}_{k+1}(G)$, $GOP^{\oplus(k+1)}(G) \restriction_\rho$ requires width $\left(\frac{1-3\epsilon}{6}\right)n$.*

9.3. The lower bound.

THEOREM 9.9. *Let k be given. There exist constants $d > 0$, $\epsilon_k > 0$ and a family of graphs G on n vertices (for n sufficiently large) with maximum degree c so that Res(k) refutations of $GOP^{\oplus(k+1)}(G)$ require size at least $2^{\Omega(\epsilon_k n)}$.*

Proof. Let k be given. Set $p = 15\ln 6/n$. Consider a random graph selected according to $G_{n,p}$; by Lemma 8.21, G is almost certainly $\frac{1}{6}$-neighborly, and by the Chernoff bounds, it has maximum degree $\leq 2pn = 26\ln 6$. Let G be a graph that is both $\frac{1}{6}$-neighborly and has maximum degree $\leq 26\ln 6$.

By Lemma 9.7, we have that for every k-DNF F, $\Pr_{\rho\in\mathcal{P}_{k+1}(G)}[F \restriction_\rho \neq 1] \leq 2^{-\delta\cdot c(F)}$. Now apply Corollary 3.4 with $s = (n/12-1)/k$, $d = 1$. For every k-DNF F,

$$\Pr_{\rho\in\mathcal{P}_{k+1}(G)}[h(F\restriction_\rho) > (n/12-1)/k] \leq k2^{-2\delta^k((n/12-1)/k)/4^k}.$$

Because k is fixed and δ depends only on k, there exists ϵ_k so that

$$\Pr_{\rho\in\mathcal{D}}[h(F\restriction_\rho) > (n/12-1)/k] < 2^{-\epsilon_k n}.$$

Suppose for the sake of contradiction that Γ is a Res(k) refutation of $GOP^{\oplus(k+1)}(G)$ of size less than $2^{\epsilon_k n}$. By the union bound, with probability greater than 0, every line F of Γ has $h(F\restriction_\rho) \leq (n/12-1)/k$. By Corollary 5.2, $GOP^{k+1}(G)\restriction_\rho$ has a resolution refutation of width at most $n/12-1$. On the other hand, because G is $\frac{1}{6}$-neighborly, $w_R(GOP(G)) \geq \left(\frac{1-3(1/6)}{6}\right)n = n/12$, and therefore $w1_R(GOP^{\oplus(k+1)}(G)\restriction_\rho) \geq w_R(GOP(G)) \geq n/12-1$. This is a contradiction. □

10. Conclusions and open problems. Switching with small restrictions seems to be a promising technique for analyzing the power of bottom fan-in in proof and circuit complexity. Our results could not have been obtained by switching with larger restrictions. For example, the lower bounds for random w-CNFs could not be proved using restrictions that set a constant fraction of the variables because some clause of the hypothesis would be falsified with high probability. Also, this method is relatively easy to apply because you do not have to re-prove the switching lemma for every lower

bound, but only check that the restrictions in question are likely to satisfy k-DNFs with high cover number.

However, switching with small restrictions still suffers from the limitations of random restriction method. In particular, it seems ineffective against random 3-CNFs and very weak pigeonhole principles. The only techniques for understanding the refutation complexity of such CNFs seem specific to resolution [8, 7, 30, 31]. Understanding the refutation complexity of these principles in Res(k) is a necessary step before understanding them in more powerful systems, and the Res(k) systems might be simple enough for the development of new techniques.

With this in mind, we suggest the following open problems as particularly relevant: (1) Do random 3-CNFs almost surely require exponential size refutations in Res(k) for all k? (2) Does there exist a family of 3-CNFs that require exponential size to refute in Res(k) but have (quasi-)polynomial-size proofs in Res($k+1$)? (3) Do Res(2) refutations of PHP_n^m require size exponential in n for all m? (4) Do there exist polynomial-size depth two Frege refutations PHP_n^{2n}? (5) Let $0 < \epsilon \leq 1/2$. Do there exist subexponential-size refutations for $PHP_n^{n+n^{1-\epsilon}}$ in Res(polylog(n))? or even in depth two Frege? (6) Does there exist a family of CNFs that require exponential-size refutations in Res(polylog(n)) but have (quasi-)polynomial-size depth two Frege refutations? (7) For given $\epsilon < \delta \leq 1$, does there exist a family of CNFs that require exponential-size refutations in Res(n^ϵ) but have (quasi-)polynomial-size Res(n^δ) refutations?

REFERENCES

[1] M. Ajtai, *Σ_1^1-formulae on finite structures*, Ann. Pure Appl. Logic, 24 (1983), pp. 1–48.

[2] A. Atserias and M. Bonet, *On the automatizability of resolution and related propositional proof systems*, in Proceedings of the Sixteenth International Workshop on Computer Science Logic, Lecture Notes in Comput. Sci. 2471, Springer, New York, 2002, pp. 569–583.

[3] A. Atserias, M. Bonet, and J. Esteban, *Lower bounds for the weak pigeonhole principle and random formulas beyond resolution*, Inform. and Comput., 176 (2002), pp. 136–152.

[4] P. Beame, *A Switching Lemma Primer*, tech. report, Department of Computer Science and Engineering, University of Washington, Seattle, 1994.

[5] P. Beame, R. Karp, T. Pitassi, and M. Saks, *The efficiency of resolution and Davis–Putnam procedures*, SIAM J. Comput., 31 (2002), pp. 1048–1075.

[6] P. Beame and T. Pitassi, *Propositional proof complexity: Past, present, and future*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 65 (1998), pp. 66–89.

[7] E. Ben-Sasson, *Hard examples for bounded depth Frege*, in Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 563–572.

[8] E. Ben-Sasson and A. Wigderson, *Short proofs are narrow—resolution made simple*, J. ACM, 48 (2001), pp. 149–169.

[9] M. Bonet and N. Galesi, *A study of proof search algorithms for resolution and polynomial calculus*, in Proceedings of the Fortieth Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alimitos, CA, 1999, pp. 422–431.

[10] R. Boppana and M. Sipser, *The complexity of finite functions*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 757–804.

[11] S. R. Buss and G. Turán, *Resolution proofs of generalized pigeonhole principles*, Theoret. Comput. Sci., 62 (1988), pp. 311–317.

[12] L. Cai, J. Chen, and J. Håstad, *Circuit bottom fan-in and computational power*, SIAM J. Comput., 27 (1998), pp. 341–355.

[13] V. Chvátal and E. Szemerédi, *Many hard examples for resolution*, J. ACM, 35 (1988), pp. 759–768.

[14] S. Cook and R. Reckhow, *The relative efficiency of propositional proof systems*, J. Symbolic Logic, 44 (1979), pp. 36–50.

[15] S. Dantchev and S. Riis, *Tree resolution proofs of the weak pigeon-hole principle*, in Proceedings of the Sixteenth Annual IEEE Conference on Computational Complexity, IEEE Computer Society Press, Los Alamitos, CA, 2001, pp. 69–75.

[16] P. Erdös and R. Rado, *Intersection theorems for finite sets*, J. London Math. Soc., 35 (1960), pp. 85–90.
[17] J. L. Esteban, N. Galesi, and J. Messner, *On the complexity of resolution with bounded conjunctions*, in Proceedings of the Twenty-Ninth International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2380, Springer, New York, 2002, pp. 220–231.
[18] M. Furst, J. Saxe, and M. Sipser, *Parity, circuits, and the polynomial-time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.
[19] A. Goerdt, *Unrestricted resolution versus N-resolution*, Theoret. Comput. Sci., 93 (1992), pp. 159–167.
[20] A. Haken, *The intractability of resolution*, Theoret. Comput. Sci., 39 (1985), pp. 297–308.
[21] J. Håstad, *Almost optimal lower bounds for small depth circuits*, in Advances in Computing Research, Vol. 5, JAI Press, Greenwich, CT, 1989, pp. 143–170.
[22] S. Jukna, *Extremal Combinatorics: With Applications to Computer Science*, Springer, New York, 2001.
[23] J. Krajíček, *Bounded Arithmetic, Propositional Logic and Complexity Theory*, Cambridge University Press, Cambridge, UK, 1995.
[24] J. Krajíček, *On the weak pigeonhole principle*, Fund. Math., 170 (2001), pp. 123–140.
[25] A. Maciel, T. Pitassi, and A. Woods, *A new proof of the weak pigeonhole principle*, J. Comput. System Sci., 64 (2002), pp. 843–872.
[26] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
[27] E. Palmer, *Graphical Evolution: An Introduction to the Theory of Random Graphs*, Wiley Interscience, Chichester, UK, 1985.
[28] T. Pitassi and R. Raz, *Regular resolution lower bounds for the weak pigeonhole principle*, in Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 347–355.
[29] P. Pudlák, *The lengths of proofs*, in Handbook of Proof Theory, S. R. Buss, ed., Elsevier North-Holland, Amsterdam, 1998, pp. 547–637.
[30] R. Raz, *Resolution lower bounds for the weak pigeonhole principle*, in Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 553–562.
[31] A. Razborov, *Improved resolution lower bounds for the weak functional pigeonhole principle*, Theoret. Comput. Sci., 303 (2001), pp. 233–243.
[32] A. Razborov, *Pseudorandom generators hard for k-DNF resolution and polynomial calculus resolution*. Available online from http://genesis.mi.ras.ru/~razborov/, 2003.
[33] N. Segerlind, S. Buss, and R. Impagliazzo, *A switching lemma for small restrictions and lower bounds for k-DNF resolution*, in Proceedings of the Forty-Third Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2002, pp. 604–613.
[34] M. Sipser, *Borel sets and circuit complexity*, in Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, ACM, New York, 1983, pp. 61–69.
[35] G. Tseitin, *On the complexity of derivations in propositional calculus*, in Studies in Constructive Mathematics and Mathematical Logic, Part II, A. O. Slisenko, ed., 1970, pp. 115–125 (in Russian).
[36] A. Urquhart, *Hard examples for resolution*, J. ACM, 34 (1987), pp. 209–219.
[37] V. Vazirani, *Approximation Algorithms*, Springer, Berlin, 2001.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1201–1214

APPROXIMATE LOCAL SEARCH IN COMBINATORIAL OPTIMIZATION*

JAMES B. ORLIN†, ABRAHAM P. PUNNEN‡, AND ANDREAS S. SCHULZ§

Abstract. Local search algorithms for combinatorial optimization problems are generally of pseudopolynomial running time, and polynomial-time algorithms are not often known for finding locally optimal solutions for NP-hard optimization problems. We introduce the concept of ε-local optimality and show that, for every $\varepsilon > 0$, an ε-local optimum can be identified in time polynomial in the problem size and $1/\varepsilon$ whenever the corresponding neighborhood can be searched in polynomial time. If the neighborhood can be searched in polynomial time for a δ-local optimum, a variation of our main algorithm produces a $(\delta+\varepsilon)$-local optimum in time polynomial in the problem size and $1/\varepsilon$. As a consequence, a combinatorial optimization problem has a fully polynomial-time approximation scheme if and only if the problem of determining a better neighbor in an exact neighborhood has a fully polynomial-time approximation scheme.

Key words. local search, neighborhood search, approximation algorithms, computational complexity, combinatorial optimization, 0/1-integer programming

AMS subject classifications. 68Q17, 68Q25, 68W25, 90C10, 90C27, 90C59

DOI. 10.1137/S0097539703431007

1. Introduction.

A *combinatorial optimization problem* Π consists of a collection of instances $(\mathcal{F}, c)$, where the set $\mathcal{F}$ of feasible solutions is a family of subsets of a finite ground set $E = \{1, \ldots, n\}$. The objective function $c : E \to \mathbb{Q}_+$ assigns a nonnegative cost to every feasible solution $S \in \mathcal{F}$ through $c(S) := \sum_{e \in S} c_e$.[1] We assume that Π is closed under componentwise scaling of objective function coefficients; i.e., if $(\mathcal{F}, c) \in \Pi$, then $(\mathcal{F}, c') \in \Pi$ for all $c' \in \mathbb{Q}_+$. For technical reasons, let us also assume that $c(S) \neq 0$ for $S \in \mathcal{F}$. The goal is to find a globally optimal solution, i.e., a feasible solution S^* such that $c(S^*) \leq c(S)$ for all $S \in \mathcal{F}$.[2] The traveling salesperson problem (TSP) or the minimum spanning tree problem are typical examples of combinatorial optimization problems (see, e.g., Lawler (1976); Papadimitriou and Steiglitz (1982); Lawler et al. (1985); Cook et al. (1998); Korte and Vygen (2002)).

Many combinatorial optimization problems are NP-hard, and one popular practical approach for attacking them is using local search strategies, which presupposes the

*Received by the editors July 8, 2003; accepted for publication (in revised form) April 27, 2004; published electronically July 20, 2004. The work of the first and third authors was partially supported by ONR contract N00014-98-1-0317. An extended abstract of this paper appeared in the Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '04), New Orleans, LA, 2004, pp. 580–589.

http://www.siam.org/journals/sicomp/33-5/43100.html

†Operations Research Center, Massachusetts Institute of Technology, Office E40-137, 77 Massachusetts Avenue, Cambridge, MA 02139-4307 (jorlin@mit.edu). The work of this author was partially supported through NSF contract DMI-0217123.

‡Department of Mathematical Sciences, University of New Brunswick, P.O. Box 5050, Saint John, New Brunswick, Canada E2L 4L5 (punnen@unbsj.ca). The work of this author was partially supported by NSERC grant OPG 0170381.

§Sloan School of Management, Massachusetts Institute of Technology, Office E53-361, 77 Massachusetts Avenue, Cambridge, MA 02139-4307 (schulz@mit.edu).

[1]Formally, we should point out that we focus on *linear* combinatorial optimization problems as opposed to "general" combinatorial optimization problems, in which the cost of a feasible solution is not necessarily the sum of the cost coefficients of its elements. In other words, the class of problems we are looking at here is equivalent to that of 0/1-integer linear programming problems.

[2]Although we restrict the following discourse to minimization problems, all results extend in a natural way to the case of maximization problems.

concept of a neighborhood. A *neighborhood function* for an instance $(\mathcal{F}, c)$ of a combinatorial optimization problem Π is a mapping $N_{\mathcal{F}} : \mathcal{F} \to 2^{\mathcal{F}}$. Note that we assume that $N_{\mathcal{F}}$ does not depend on the objective function c. For convenience, we usually drop the subscript and simply write N. For a feasible solution S, $N(S)$ is called the *neighborhood* of S. We assume that $S \in N(S)$. A feasible solution $\bar{S}$ is said to be *locally optimal* with respect to N if $c(\bar{S}) \leq c(S)$ for all $S \in N(\bar{S})$. The *local search problem* is that of finding a locally optimal solution. Classic neighborhood functions include the k-opt neighborhood for the TSP (Lin (1965)), the flip neighborhood for MAX CUT and MAX 2SAT (Schäffer and Yannakakis (1991)), and the swap neighborhood for the graph partitioning problem (Kernighan and Lin (1970)). However, the class of problems that we are considering here also includes neighborhoods of exponential size, such as, for the TSP, the twisted sequences neighborhood, the pyramidal tours neighborhood, the permutation tree neighborhood, neighborhoods based on partial orders, and neighborhoods induced by polynomial-time solvable special cases. We refer the reader to Deineko and Woeginger (2000), Ahuja et al. (2002), and Gutin, Yeo, and Zverovitch (2002) for a detailed description of these neighborhood functions.

Roughly speaking, a local search algorithm sets out with an initial feasible solution and then repeatedly searches neighborhoods to find better and better solutions until it reaches a locally optimal solution. Figure 1.1 gives a generic description of the standard local search algorithm, which is sometimes also called *iterative improvement*.

Step 1: Compute a feasible starting solution $\bar{S}$;
Step 2: **while** $\bar{S}$ is not locally optimal **do**
Choose $S \in N(\bar{S})$ such that $c(S) < c(\bar{S})$;
$\bar{S} := S$;
Step 3: Output $\bar{S}$.

FIG. 1.1. *Standard local search algorithm.*

Computational studies of local search algorithms and their variations have been extensively reported in the literature for various combinatorial optimization problems (see, e.g., Johnson et al. (1989) and Johnson and McGeoch (1997) for studies of the graph partitioning problem and the TSP, respectively). Empirically, local search heuristics appear to converge rather quickly, within low-order polynomial time. Compared to this wealth of information on empirical analysis, relatively little is known about theoretical properties of this class of algorithms. Of course, if one first multiplies all cost coefficients with their smallest common denominator to make them integer, the standard local search algorithm terminates in a pseudopolynomial number of iterations since it improves the objective function value by an integral amount in each iteration.[3] However, polynomial-time algorithms for computing a local optimum are in general not known. This is especially true for the above-mentioned combinatorial optimization problems and neighborhoods. On the other hand, Lawler (1976) constructed instances of the metric TSP such that the standard local search algorithm with the 2-opt neighborhood takes an exponential number of iterations under a particular pivoting rule. Chandra, Karloff, and Tovey (1999) extended this result to k-opt for all fixed $k > 2$.

[3] Note that the scaling of rational coefficients to integers does not cause a superpolynomial blowup. We henceforth assume without loss of generality (w.l.o.g.) that all cost coefficients c_e for $e \in E$ are integers. An algorithm is pseudopolynomial if it is polynomial in the input dimension n and in $c_{\max} := \max_{e \in E} c_e$.

Input:	Objective function $c : E \to \mathbb{N}$ and feasible solution $S \in \mathcal{F}$.
Output:	"YES" if S is locally optimal with respect to c and N.
	"NO" and S' if there exists $S' \in N(S)$ with $c(S') < c(S)$.

FIG. 1.2. *Specification of the subroutine (oracle)* IMPROVE$_N$.

This discontenting situation prompted Johnson, Papadimitriou, and Yannakakis (1988) to introduce the complexity class PLS (for polynomial-time local search). A combinatorial optimization problem Π together with a given neighborhood function N belongs to PLS if (a) instances are polynomial-time recognizable and a feasible solution is efficiently computable, (b) the feasibility of a proposed solution can be checked in polynomial time, and (c) neighborhoods can be searched efficiently. That is, there is a polynomial-time algorithm that decides whether a given feasible solution is locally optimal and, if not, computes a better solution in its neighborhood; see Figure 1.2 for a detailed description of the input and output of this algorithm. Note that all common local search problems are in PLS, in particular the problems mentioned earlier. The class PLS has its own type of reduction, which gives rise to the identification of complete problems in that class. For instance, MAX CUT and MAX 2SAT with the flip neighborhood, graph partitioning with the swap neighborhood, and TSP with the Lin–Kernighan neighborhood are PLS-complete (Krentel (1990); Schäffer and Yannakakis (1991); Papadimitriou (1992)), and so is TSP with the k-opt neighborhood for some constant k (Krentel (1989)). In particular, if a local optimum can be found in polynomial time for one of these problems, then a local optimum can be computed in polynomial time for all problems in PLS. Unfortunately, it is unknown whether it is hard to find a local optimum for a PLS-complete problem (but Johnson, Papadimitriou, and Yannakakis (1988) pointed out that this would imply NP = co-NP) or whether this can be done in polynomial time.[4]

In light of this somewhat elusive situation, it is interesting to explore the possibility of identifying *approximately locally optimal* solutions in polynomial time. We therefore introduce the notion of an ε-locally optimal solution, which is to some extent related to the worst-case relative performance guarantee of an approximation algorithm. We say that a feasible solution S^ε to an instance of a combinatorial optimization problem Π with neighborhood function N is an *ε-local optimum* if

$$\frac{c(S^\varepsilon) - c(S)}{c(S)} \leq \varepsilon \qquad \text{for all } S \in N(S^\varepsilon)$$

for some $\varepsilon > 0$. Hence, while S^ε is not necessarily a local optimum, it "almost" is. A family of algorithms $(A_\varepsilon)_{\varepsilon>0}$ for Π is an *ε-local optimization scheme* if A_ε produces an ε-local optimum. If the running time of algorithm A_ε is polynomial in the input size and $1/\varepsilon$, it is called a *fully polynomial-time ε-local optimization scheme*. In this paper, we show that every combinatorial optimization problem with an efficiently searchable neighborhood has a fully polynomial-time ε-local optimization scheme. In particular, an ε-locally optimal solution can be computed in polynomial time for every problem

[4]Note that the negative results for the TSP mentioned at the end of the previous paragraph apply only to the standard local search algorithm, as described in Figure 1.1. Actually, there exist instances for every PLS-complete problem for which the standard local search algorithm takes exponential time, regardless of the tie-breaking and pivoting rules used (Yannakakis (1997), Theorem 13).

in PLS, including the PLS-complete problems and the problems with exponentially sized neighborhoods mentioned above.

Related work. Ausiello and Protasi (1995) introduced the class GLO (for guaranteed local optima) of optimization problems that have the property that the objective function value of each local optimum is guaranteed to be within a constant factor of that of a global optimum. Khanna et al. (1998) extended this notion to nonoblivious GLO problems, which allow for a modification of the objective function used to compute a local optimum. In either class, the underlying neighborhoods contain all solutions of bounded Hamming distance from the current solution; moreover, it is assumed that the number of distinct objective function values over the set of feasible solutions is polynomially bounded. Hence, the standard local search algorithm yields a locally optimal solution in polynomial time. In contrast, we do not make any assumption on the objective function values or neighborhoods considered; we show that an ε-local optimum can always be computed with a polynomial number of calls to the IMPROVE subroutine. On the other hand, while an ε-local optimum has nearly the properties of a local optimum, its objective function value is not guaranteed to be close to that of a global optimum. However, this is in general true for local optima as well. For instance, Papadimitriou and Steiglitz (1977) showed that no local optimum of an efficiently searchable neighborhood for the TSP can be within a constant factor of the optimal value unless P = NP. Yet, whenever a combinatorial optimization problem has an efficiently searchable neighborhood such that the value of each local optimum is within a constant factor $\alpha \geq 1$ of that of a global minimum, then we can compute in polynomial time an ε-local optimum of cost not worse than $\alpha + \varepsilon$ times that of a global optimum. In other words, we give an $(\alpha+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$.

Klauck (1996) studied the complexity of finding a solution whose objective function value is approximately as good as that of the worst local optimum by using a restricted form of PLS-reductions. Completeness under this reduction implies that an approximation of a local optimum cannot be achieved efficiently unless P = PLS. For instance, 0/1-programming with the k-flip neighborhood and the TSP with the k-opt neighborhood for constant k are complete under this reduction.

A neighborhood function N of a combinatorial optimization problem Π is *exact* if every locally optimal solution with respect to N is also globally optimal. In this case, our fully polynomial-time ε-local optimization scheme actually is a fully polynomial-time approximation scheme (FPTAS). This remains true even if IMPROVE already outputs "YES" when the current feasible solution is a δ-local optimum (and does so in time polynomial in the input size and $1/\delta$ for any $\delta > 0$). Grötschel and Lovász (1995) and Schulz, Weismantel, and Ziegler (1995) showed that if a combinatorial optimization problem has an exact neighborhood that can be searched efficiently (and exactly), one can actually find an exact optimal solution efficiently. Schulz and Weismantel (1999, 2002) discussed extensions of this result from 0/1-integer linear programming problems (i.e., combinatorial optimization problems) to arbitrary integer programs. However, none of the employed techniques can be extended to compute a local optimum in polynomial time unless the neighborhood is exact. In fact, otherwise P = PLS.

Fischer (1995) examined a different question, which implies that our main result is best possible if one considers the class of algorithms that iteratively move from one feasible solution to a feasible solution in its neighborhood: Given a feasible solution S to an instance $(\mathcal{F}, c)$ of a combinatorial optimization problem Π and a number k

in unary, does there exist a local optimum within k neighborhood steps of S? She showed that this question is NP-complete for MAX CUT and MAX 2SAT under the flip neighborhood and for the TSP under the 2-opt neighborhood, among others.

Our results. Apart from our main result presented above and discussed in more detail in section 2 below, we offer various evidence in section 3 to show that this result is indeed the "best possible."

First, the fully polynomial-time ε-local optimization scheme produces an ε-locally optimal solution by proceeding from one feasible solution to a solution in its neighborhood, and so forth. In this sense, it is a typical local search algorithm. Fischer's result shows that one cannot hope to find a local optimum in polynomial time by proceeding in this manner. Yet, an ε-local optimum can be determined in polynomial time. The key is to modify the original objective function so as to make sufficient progress in a relatively small number of steps. It is worth mentioning that our algorithm follows a path of feasible solutions that is not necessarily monotone with respect to the original objective function. In particular, it differs from a standard local search algorithm, which always replaces the current solution with a neighboring solution of lower cost.

Second, we point out that *any* algorithm for computing a local optimum that treats the neighborhood search and the feasibility check of PLS problems as oracles must be in the worst case exponential in the input dimension. In other words, if there exists a polynomial-time algorithm to compute a local optimum for a PLS-complete problem, it must use problem-specific knowledge. In contrast, our algorithmic scheme works for any combinatorial optimization problem so long as a feasible solution can be efficiently computed; in particular, it treats the subroutine IMPROVE as a black box.

Third, we show that the existence of a family $(A_\varepsilon)_{\varepsilon>0}$ of algorithms that find ε-local optima in time polynomial in the input size and $\log 1/\varepsilon$ implies the existence of a polynomial-time algorithm for computing a local optimum, and we just argued why this is impossible in our framework. Hence, the dependence of the running time on $1/\varepsilon$ cannot be significantly improved. Furthermore, we prove that replacing the relative error in the definition of an ε-local optimum with the absolute error would also yield the existence of a polynomial-time algorithm for computing a local optimum.

Finally, in section 4 we present various extensions and variations of the main result, including more general integer linear programming problems and the new characterization for when a combinatorial optimization problem has an FPTAS, which we already described in the context of related results for exact neighborhoods. Moreover, we discuss neighborhoods of polynomial size and efficiently searchable neighborhoods with local optima guaranteed to be near-optimal.

2. A fully polynomial-time ε-local optimization scheme. In this section we develop a polynomial-time algorithm to compute an ε-local optimum for a given instance $(\mathcal{F}, c)$ with ground set E of a combinatorial optimization problem Π with neighborhood function N.[5] The algorithm starts with a feasible solution S^0. We then alter the element costs c_e for $e \in E$ according to a prescribed scaling rule to generate a modified instance. Using local search on this modified problem, we look for a solution with an objective function value (with respect to the original cost) that is half that of S^0. If no such solution is found, we are at a local optimum for the

[5] The algorithm presented here works for neighborhoods of any size, in particular exponential-sized neighborhoods. Section 4.3 features a somewhat simpler algorithm for neighborhoods of polynomial size, which are given explicitly.

Input: Objective function $c : E \to \mathbb{N}$; subroutine IMPROVE$_N$; initial feasible solution $S^0 \in \mathcal{F}$; accuracy $\varepsilon > 0$.

Output: Solution $S^\varepsilon \in \mathcal{F}$ that is an ε-local optimum with respect to N and c.

Step 1: Let $i := 0$;

Step 2: Let $K := c(S^i)$, $q := \dfrac{K\varepsilon}{2n(1+\varepsilon)}$, and $c'_e := \left\lceil \dfrac{c_e}{q} \right\rceil q$ for $e \in E$;

Step 3: Let $k := 0$ and $S^{i,k} := S^i$;

Step 4: **repeat**

Call IMPROVE$_N(S^{i,k}, c')$;

if the answer is "NO," **then**

Let $S^{i,k+1} \in N(S^{i,k})$ such that $c'(S^{i,k+1}) < c'(S^{i,k})$; set $k := k + 1$;

else $S^\varepsilon := S^{i,k}$; **stop**.

until $c(S^{i,k}) \le K/2$;

Step 5: Let $S^{i+1} := S^{i,k}$, set $i := i + 1$ and **goto** Step 2.

FIG. 2.1. *ε-local search algorithm.*

modified problem and output this solution. Otherwise we replace S^0 by the solution of cost less than half, we call the latter one S^1, and the algorithm is repeated. A formal description of the algorithm is given in Figure 2.1. Note that the modification of the cost coefficients in Step 2 merely amounts to rounding them up to the closest integer multiple of q. Let us establish correctness first.

THEOREM 2.1. *The ε-local search algorithm produces an ε-local optimum.*

Proof. The ε-local search algorithm terminates, which follows from the running time analysis following this proof. Let S^ε be the solution produced by the algorithm, and let $S \in N(S^\varepsilon)$ be an arbitrary solution in its neighborhood. Let K and q denote the corresponding values from the last execution of Step 2 of the algorithm. Note that

$$c(S^\varepsilon) = \sum_{e \in S^\varepsilon} c_e \le \sum_{e \in S^\varepsilon} \left\lceil \frac{c_e}{q} \right\rceil q \le \sum_{e \in S} \left\lceil \frac{c_e}{q} \right\rceil q \le \sum_{e \in S} q\left(\frac{c_e}{q} + 1 \right) \le \sum_{e \in S} c_e + nq = c(S) + nq,$$

where $n = |E|$. Here, the second inequality follows from the fact that S^ε is locally optimal with respect to c'. Together with $c(S^\varepsilon) \ge K/2$, this implies

$$\frac{c(S^\varepsilon) - c(S)}{c(S)} \le \frac{nq}{c(S)} \le \frac{nq}{c(S^\varepsilon) - nq} \le \frac{2nq}{K - 2nq} = \varepsilon. \qquad \square$$

Let us now analyze the running time of the ε-local search algorithm. In each improving move within the local search in Step 4 of the algorithm, the objective function value (with respect to c') is decreased by at least q units. Thus the number of calls to IMPROVE between two consecutive iterations of Step 2 is $O(n(1+\varepsilon)/\varepsilon) = O(n/\varepsilon)$. Step 2 is executed at most $\log c(S^0)$ times, where S^0 is the starting solution. Thus the total number of neighborhoods searched is $O(n\,\varepsilon^{-1} \log c(S^0))$. Therefore, if the neighborhood N can be searched in polynomial time for an improving solution, we have a fully polynomial-time ε-local optimization scheme. Note that the number of

iterations includes the factor $\log c(S^0)$, and hence the bound is not strongly polynomial. However, it is possible to prove a strongly polynomial bound on the number of iterations. For this, we make use of the following lemma, which Radzik (1993) attributed to Goemans.

LEMMA 2.2. *Let $d = (d_1, \ldots, d_n)$ be a real vector and let $y_1, \ldots, y_p$ be vectors in $\{0,1\}^n$. If, for all $i = 1, \ldots, p-1$, $0 \le d\,y_{i+1} \le \frac{1}{2}\,d\,y_i$, then $p = \mathrm{O}(n \log n)$.*

Note that the value of K at each execution of Step 2 is reduced at least by half. Further, K is a linear combination of c_e for $e \in E$ and the coefficients in this linear combination are from the set $\{0,1\}$. Lemma 2.2 implies that Step 2 of the ε-local search algorithm can be executed at most $\mathrm{O}(n \log n)$ times. Thus the total number of calls of IMPROVE in Step 4 throughout the algorithm is $\mathrm{O}(\varepsilon^{-1} n^2 \log n)$. If $\zeta(n, \log c_{\max})$ is the time needed to search the neighborhood N for an improving solution (i.e., the running time of IMPROVE) and $\xi(n)$ is the time needed to obtain a feasible starting solution, the complexity of the ε-local search algorithm is $\mathrm{O}(\xi(n) + \zeta(n, \log c_{\max}) n\,\varepsilon^{-1} \min\{n \log n, \log K^0\})$, where $K^0 := c(S^0)$ is the objective function value of the starting solution and $c_{\max}$ is the maximal value of an objective function coefficient. The following theorem summarizes the preceding discussion.

THEOREM 2.3. *The ε-local search algorithm correctly identifies an ε-locally optimal solution of an instance of a combinatorial optimization problem in $\mathrm{O}(\xi(n) + \zeta(n, \log c_{\max}) n\,\varepsilon^{-1} \min\{n \log n, \log K^0\})$ time.*

Thus if $\zeta(n, \log c_{\max})$ and $\xi(n)$ are polynomial, then the ε-local search algorithm is a fully polynomial-time ε-local optimization scheme. Note that $\zeta(n, \log c_{\max})$ and $\xi(n)$ are indeed polynomials of the input size for all problems in PLS.

COROLLARY 2.4. *Every problem in PLS has a fully polynomial-time ε-local optimization scheme.*

The running time of the ε-local search algorithm can sometimes be improved by exploiting the special structure of the underlying neighborhood. A neighborhood function N generates a so-called k-opt neighborhood if $S_1, S_2 \in N(S)$ implies $|(S_1 \setminus S_2) \cup (S_2 \setminus S_1)| \le k$, which is equivalent to bounding the Hamming distance between the incidence vectors of S_1 and S_2 by k. For the k-opt neighborhood, by choosing the parameter $q := \frac{K\varepsilon}{2k(1+\varepsilon)}$ in the ε-local search algorithm, we still get an ε-local optimum. Moreover, the number of calls of IMPROVE between two consecutive executions of Step 2 of this modified algorithm is $\mathrm{O}(\varepsilon^{-1})$ for fixed k. This brings down the total number of such calls to $\mathrm{O}(\varepsilon^{-1} \min\{n \log n, \log K^0\})$ and implies a running time of $\mathrm{O}(\xi(n) + n^k \varepsilon^{-1} \min\{n \log n, \log K^0\})$ for the ε-local search algorithm.

Similarly, if the considered combinatorial optimization problem possesses a 2-approximation algorithm, one can use this algorithm to compute the starting solution S^0. If such a solution is used as the starting solution, then the ε-local search algorithm executes Step 2 only once and hence the total number of improving moves is $\mathrm{O}(n\,\varepsilon^{-1})$. Consequently, the overall running time of the ε-local search algorithm is $\mathrm{O}(\xi(n) + \zeta(n, \log c_{\max}) n\,\varepsilon^{-1})$, where $\xi(n)$ now denotes the running time of the 2-approximation algorithm. In fact, whenever one has a feasible solution S^0 for an instance I such that $c(S^0) \le p(\langle I \rangle) c(S^*)$ for some polynomial p of the input size $\langle I \rangle$, then one can adapt the value of q to $q := (K\varepsilon)/(n\,p(\langle I \rangle)(1+\varepsilon))$ and the stopping criterion of the main loop accordingly, so that the ε-local search algorithm computes an ε-local optimum in $\mathrm{O}(n\,p(\langle I \rangle)\varepsilon^{-1})$ iterations.

Arkin and Hassin (1998) applied a similar approach to that used in the ε-local search algorithm to compute a local optimum in polynomial time in the context of the weighted k-set packing problem. They considered a neighborhood for which the value of each local optimum is within a certain factor of the value of a global optimum;

hence, this approach, which they attributed to Rubinstein, leads to a polynomial-time approximation algorithm. It has since been applied in related situations as well; see, e.g., Arkin et al. (2002).

3. Negative results. In this section we present a collection of results that underscore that neither the accuracy of the solutions produced by the ε-local search algorithm nor its running time can be significantly improved unless additional, problem-specific knowledge is used. Let us first argue that any algorithm to compute a local optimum for a problem in PLS has to have exponential running time in the worst case if the algorithms for checking feasibility and neighborhood search are oracles completely hiding both the set of feasible solutions and the neighborhood structure.

THEOREM 3.1. *If the only available information on an instance $(\mathcal{F}, c)$ of a combinatorial optimization problem Π is the objective function vector c, a feasible solution $S^0 \in \mathcal{F}$, a membership oracle, and a neighborhood search oracle* IMPROVE*, then any algorithm for computing a local optimum takes exponential time in the worst case.*

Proof. Let the ground set be $E = \{1, 2, \ldots, n\}$. The objective function coefficients are $c_i = 2^{i-1}$ for $i = 1, 2, \ldots, n$. Let the nonempty subsets of E be labeled $S^0, S^1, \ldots, S^{2^n-2}$ such that $c(S^0) > c(S^1) > \cdots > c(S^{2^n-2})$. Let A be an arbitrary algorithm that computes a local optimum. Note that A can either call IMPROVE with a feasible solution and some objective function or check whether a particular solution is feasible by asking the membership oracle. We show that A needs exponential time in the worst case by exhibiting an adverse strategy. In fact, an adversary adapts the set of feasible solutions and the neighborhood function to A's sequence of questions as follows. Whenever A asks the membership oracle whether a set S^i is feasible, the adversary answers "NO" unless S^i has been declared feasible earlier. On the other hand, whenever A calls up IMPROVE with a feasible solution S^i and an objective function vector c', IMPROVE returns S^j, where $j > i$ is the smallest index for which S^j has not been labeled infeasible. If no such j exists or $c'(S^j) \geq c'(S^i)$, then S^i is locally optimal (with respect to c'). It is not difficult to see that A has to touch every single subset before it can identify the unique minimum. □

The importance of Theorem 3.1 relates to the fact that the ε-local search algorithm requires only a subset of the information stated in the assumptions of this theorem; in particular, it does not make use of the membership oracle.

We next note that finding an ε-local optimum of *additive error* ε with respect to a given neighborhood structure is as hard as finding a local optimum with respect to the same neighborhood structure. While its proof relies on a standard argument, the result is still worth recording.

OBSERVATION 3.2. *If there is an algorithm that for every instance $(\mathcal{F}, c)$ of a combinatorial optimization problem Π with neighborhood N finds in polynomial time a feasible solution S_ε such that $c(S_\varepsilon) \leq c(S) + \varepsilon$ for all $S \in N(S_\varepsilon)$ for some fixed $\varepsilon > 0$, then there is a polynomial-time algorithm to find a local optimum.*

Proof. Let $(\mathcal{F}, c)$ be an instance of Π, where w.l.o.g. c is an integer-valued function. Create a new instance $(\mathcal{F}, c')$ by setting $c'_e := (1+\varepsilon)c_e$ for all elements e of the ground set. Apply the given algorithm to the new instance and let S' be the resulting solution. Then $c'(S') - c'(S) \leq \varepsilon$ for all $S \in N(S')$. Thus, $c(S') - c(S) \leq \varepsilon/(\varepsilon + 1) < 1$ for all $S \in N(S')$. Since c is integer-valued, it follows that S' is a local optimum for the original instance. □

The next result is somewhat similar to (Garey and Johnson (1979), Theorem 6.8) except that we are discussing it in the context of local optimality.

OBSERVATION 3.3. *If a combinatorial optimization problem Π has a fully poly-*

nomial-time ε-local optimization scheme $(A_\varepsilon)_{\varepsilon>0}$ such that the actual running time of A_ε is polynomial in the input size and $\log 1/\varepsilon$, then there is a polynomial-time algorithm that computes a local optimum.

Proof. Let $(\mathcal{F}, c)$ be an instance of Π, where w.l.o.g. c is an integer-valued function. Choose $\varepsilon := 1/(n\, c_{\max} + 1)$ and apply A_ε. Note that its running time is polynomial in the input size of the instance. If S^ε is the solution returned by this algorithm, then $c(S^\varepsilon) \le (1+\varepsilon)c(S) < c(S) + 1$ for all $S \in N(S^\varepsilon)$. Hence, S^ε is a local optimum. ∎

4. Extensions and variants. We now discuss some extensions and variations of the results of section 2 that range from approximation guarantees in the case of exact neighborhoods, over simplifications of the ε-local search algorithm for explicitly given neighborhoods of polynomial size, to replacing IMPROVE with weaker oracles and bounded integer programming problems.

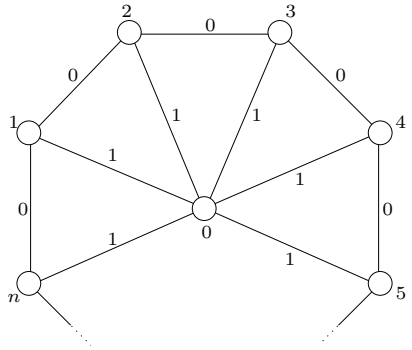


FIG. 4.1. *A wheel on $n+1$ nodes.*

4.1. Exact neighborhoods. Recall that a neighborhood function N for a combinatorial optimization problem Π is exact if every local optimum is already globally optimal. In view of this, one may be tempted to conjecture that the objective function value of an ε-local optimum with respect to an exact neighborhood is also within a factor of $(1+\varepsilon)$ of the value of a global optimum. However, this is not true as shown by the following example.

Let $G = (V, E)$ be a connected graph with edge weights c_e for $e \in E$. Let $\mathcal{F}$ be the family of all spanning trees of G. Hence, we are considering the minimum spanning tree problem. For any tree $T \in \mathcal{F}$, consider the neighborhood $N(T)$ that consists of those spanning trees obtained from T by adding an edge $e \in E \setminus T$ to T and removing an edge $f \in T$ from the induced elementary cycle. This is the 2-opt neighborhood, which is known to be exact. Now choose G as a wheel (see Figure 4.1) with node set $\{0, 1, \ldots, n\}$. For each edge $(0, i)$, $i = 1, 2, \ldots, n$, of this wheel, assign a cost of 1, and for each edge $(i, i+1)$, $i = 1, 2, \ldots, n$ (where node $n+1$ is identified with node 1), assign a cost of zero. The spanning tree T^ε, which is a star rooted at node 0, is a $1/(n-1)$-local optimum for any $n \ge 3$. However, the Hamiltonian path $T^* = (0, 1, \ldots, n)$ is a minimum spanning tree and $c(T^\varepsilon) - c(T^*) = (n-1)c(T^*)$. Thus, T^ε is not a $(1+\varepsilon)$-approximation for any $\varepsilon < n - 1$.

Still, for exact neighborhoods our fully polynomial-time ε-local optimization scheme actually is an FPTAS, as the following theorem shows.

THEOREM 4.1. *If the neighborhood N of a combinatorial optimization problem Π is exact, then the objective function value of the solution produced by the ε-local search algorithm is within a factor of $(1+\varepsilon)$ of that of a global minimum.*

Proof. Let $(\mathcal{F}, c)$ be a given instance of Π. Let S^* be an optimal solution and let S^ε be the solution produced by the algorithm. Let K, q, and c' denote the corresponding values from the last execution of Step 2 of the ε-local search algorithm. Since S^ε is locally optimal with respect to c' and the neighborhood is exact, S^ε is an optimal solution for $(\mathcal{F}, c')$. Thus,

$$c(S^\varepsilon) \le \sum_{e \in S^\varepsilon} \left\lceil \frac{c_e}{q} \right\rceil q \le \sum_{e \in S^*} \left\lceil \frac{c_e}{q} \right\rceil q \le \sum_{e \in S^*} q\left(\frac{c_e}{q} + 1\right) \le c(S^*) + nq \le c(S^*) + \frac{\varepsilon}{1+\varepsilon}\, c(S^\varepsilon),$$

where the last inequality follows from the definition of q and the fact that $c(S^\varepsilon) \ge K/2$. The result follows. □

Theorem 4.1 implies that whenever a combinatorial optimization problem has an exact neighborhood and an initial feasible solution is readily available, then the ε-local search algorithm is a $(1+\varepsilon)$-approximation algorithm that calls IMPROVE a polynomial number of times. However, Grötschel and Lovász (1995) and Schulz, Weismantel, and Ziegler (1995) showed that under these circumstances, one can actually compute an optimal solution with a polynomial number of calls of IMPROVE. Yet, one still obtains an FPTAS even if the exact neighborhood can only be searched approximately, as we are about to see next.

4.2. Approximate version of neighborhood search. Very large-scale neighborhood (VLSN) search algorithms are local search algorithms using neighborhoods of very large size; see Ahuja et al. (2002) for a survey. For many VLSNs of NP-hard combinatorial optimization problems, the problem of finding an improving solution is itself NP-hard. On the other hand, solutions produced by local search algorithms using such huge neighborhoods could be of very high quality. To keep the complexity of a VLSN algorithm manageable, approximation algorithms are often employed to search an underlying neighborhood such that if the approximation algorithm fails to find an improving move, the algorithm terminates, leaving an "approximate" local solution. More precisely, instead of IMPROVE, we may only have at our command a subroutine δ-IMPROVE, which solves the following problem:

(4.1)

Given an objective function vector c and a solution $S \in \mathcal{F}$, find $S' \in N(S)$ such that $c(S') < c(S)$, or assert that S is a δ-local optimum.

Naturally, finding a δ-local optimum efficiently, given an algorithm δ-IMPROVE, faces similar difficulties as the problem of finding a local optimum when an algorithm IMPROVE is provided. However, one can easily modify the ε-local search algorithm so that it computes a $(\delta+\varepsilon)$-local optimum in polynomial time. In fact, if one uses δ-IMPROVE in lieu of IMPROVE and selects the scaling parameter q to be $q := \frac{K\varepsilon}{2n(1+\delta)(1+\delta+\varepsilon)}$, the resulting algorithm produces a $(\delta+\varepsilon)$-local optimum in time $O(\xi(n) + \psi(n, \log c_{\max})\, n\, \varepsilon^{-1} \min\{n \log n, \log K^0\})$, where $\psi(n, \log c_{\max})$ is the running time of δ-IMPROVE. Hence, a (strongly) polynomial-time algorithm for solving (4.1) implies a (strongly) polynomial-time algorithm to compute a $(\delta+\varepsilon)$-local optimum for every $\varepsilon > 0$. In view of the results discussed in section 4.1, it is particularly interesting to note that in the case of an exact neighborhood, the existence of a polynomial-time algorithm δ-IMPROVE for all $\delta > 0$ implies that the combinatorial optimization problem possesses an FPTAS. Indeed, if we call (4.1) the *augmentation problem* if N is exact (e.g., $N(S) = \mathcal{F}$ for all $S \in \mathcal{F}$) and $\delta = 0$, and a family of algorithms δ-IMPROVE (with running time polynomial in the input size and $1/\delta$ for all

$\delta > 0$) a *fully polynomial-time approximation scheme for the augmentation problem*, we can state the following result.

THEOREM 4.2. *A combinatorial optimization problem has a fully (strongly) polynomial-time approximation scheme if and only if its corresponding augmentation problem has a fully (strongly) polynomial-time approximation scheme.*

The proof of Theorem 4.2 is similar to that of Theorem 4.1.

Sometimes the objective function value of each local optimum is within a constant factor of that of a global optimum. The class of problems with this property contains the class GLO (Ausiello and Protasi (1995)). For instance, each local optimum of the flip neighborhood for the MAX CUT problem has value at least half that of an optimal cut (Sahni and Gonzalez (1976)). The following theorem can be proved in a similar manner to Theorem 4.1 by setting $q := \frac{K\varepsilon}{2n\alpha(1+\varepsilon)}$ in the ε-local search algorithm.

THEOREM 4.3. *Let Π be a combinatorial optimization problem with efficiently searchable neighborhood function N such that every local optimum is within a constant factor $\alpha \geq 1$ of the optimal value. Then there is an algorithm that computes a feasible solution of cost at most $\alpha + \varepsilon$ times that of an optimal solution in time polynomial in the input size and $1/\varepsilon$, for any $\varepsilon > 0$.*

4.3. Polynomial-sized neighborhoods. The ε-local search algorithm is designed to work for any local search problem for which an IMPROVE (or δ-IMPROVE) oracle is available, regardless of the size or structure of the neighborhood and the implementation of IMPROVE. In particular, the ε-local search algorithm identifies for the TSP and each of the following neighborhoods an ε-local optimum in polynomial time: the twisted sequences neighborhood, the pyramidal tours neighborhood, the permutation tree neighborhood, neighborhoods based on partial orders, as well as neighborhoods induced by polynomial-time solvable special cases. While all these exponential-sized neighborhoods can be searched efficiently (i.e., they have polynomial-time improvement oracles), it is not known for any of them how to find a local optimum in polynomial time (Deineko and Woeginger (2000); Ahuja et al. (2002); Gutin, Yeo, and Zverovitch (2002)).

Input:	Objective function $c : E \to \mathbb{N}$; neighborhood function $N : \mathcal{F} \to 2^{\mathcal{F}}$; initial feasible solution $S \in \mathcal{F}$; accuracy $\varepsilon > 0$.
Output:	Solution $S^{\varepsilon} \in \mathcal{F}$ that is an ε-local optimum with respect to N and c.
Step 1:	**while** S is not ε-locally optimal **do** Choose $S' \in N(S)$ satisfying $c(S') < c(S)/(1+\varepsilon)$; $S := S'$;
Step 2:	**return** $S^{\varepsilon} := S$.

FIG. 4.2. *ε-local search algorithm for neighborhoods of polynomial size.*

Nonetheless, neighborhoods frequently are of polynomial size and explicitly given, like the k-opt neighborhood for the TSP (for fixed k), the flip neighborhood for MAX CUT and MAX 2SAT, or the swap neighborhood for the graph partitioning problem. In this case, one can give a simpler algorithm for computing an ε-local optimum; see Figure 4.2. Note that Step 1 can be realized by an exhaustive search of the neighborhood of S. Obviously, the running time of this algorithm is polynomial in the input size and $1/\varepsilon$. By using an appropriately modified version of Lemma 2.2, one can actually show that the running time depends only on the input dimension n and $1/\varepsilon$.

However, this simpler algorithm has some drawbacks compared to the general ε-local search algorithm, even if one limits this comparison to problems with explicitly given neighborhoods of polynomial size. In particular, neither Theorem 4.1 nor Theorem 4.2 can be proved with its help. In fact, for the minimum spanning tree example described in Figure 4.1, the simpler algorithm with accuracy $\varepsilon \geq 1/(n-1)$ would terminate with the initially given solution T^{ε}, although the cost of this solution is n times that of an optimal solution. In contrast, the original ε-local search algorithm would return an optimal solution in this case (and a $(1+\varepsilon)$-approximate solution in general).

4.4. Weaker version of neighborhood search. In the context of global optimization, Schulz, Weismantel, and Ziegler (1997) pointed out that the requirements on the improvement oracle can be somewhat weakened. Interestingly, this extension also works in the context of local optimization, as we will show now. For that, we replace IMPROVE with another subroutine, which we call TEST. TEST accepts the same input as IMPROVE, namely, a current feasible solution S together with an objective function vector c. It also answers "YES" or "NO" depending on whether S is locally optimal with respect to c or not, but in contrast to IMPROVE it does not provide a solution $S' \in N(S)$ of lower cost if S is not locally optimal. It just answers "YES" or "NO."

LEMMA 4.4. TEST *can emulate* IMPROVE *in polynomial time; i.e., whenever the input solution S is not locally optimal, a polynomial number of calls to* TEST *suffices to create a solution $S' \in N(S)$ of lower cost.*[6]

Proof. Let $(\mathcal{F}, c)$ be an instance and S a feasible solution that is not locally optimal with respect to c under the given neighborhood N. W.l.o.g., we may assume that $S = E$.[7] Here, $E = \{1, 2, \ldots, n\}$ is the ground set. The algorithm that we are about to explain proceeds by considering one coordinate after the other. In particular, it will call TEST n times. The first call $\text{TEST}_N(S, c^1)$ is made with the objective function

$$c_e^1 := \begin{cases} c_1 - M & \text{if } e = 1, \\ c_e & \text{otherwise} \end{cases} \qquad \text{for } e \in E,$$

where $M := n\, c_{\max} + 1$. If TEST responds with "YES," then we can infer that all solutions S' in $N(S)$ of cost lower than S with respect to c satisfy $1 \notin S'$. On the other hand, if the reply is "NO," then there is at least one solution $S' \in N(S)$ such that $c(S') < c(S)$ and $1 \in S'$.

In general, assume that we already know a subset $R \subseteq \{1, 2, \ldots, k\}$ for some $k \in \{1, 2, \ldots, n-1\}$ with the following two properties:

(i) there exists a solution $S' \in N(S)$ with $c(S') < c(S)$ such that $R = S' \cap \{1, 2, \ldots, k\}$;
(ii) if $j \notin R$ for $1 \leq j \leq k$, then all solutions $S'' \in N(S)$ with $c(S'') < c(S)$ satisfy $R \cap \{1, 2, \ldots, j\} = S'' \cap \{1, 2, \ldots, j\}$.

We then call $\text{TEST}_N(S, c^{k+1})$ with

$$c_e^{k+1} := \begin{cases} c_e - M & \text{if } e \in R, \\ c_{k+1} - M & \text{if } e = k+1, \\ c_e & \text{otherwise} \end{cases} \qquad \text{for } e \in E.$$

If the result is "YES," then we can infer that all solutions $S' \in N(S)$ with $c(S') < c(S)$ and $S' \cap \{1, 2, \ldots, k\} = R$ satisfy $k+1 \notin S'$. However, if the reply is "NO," then there

[6] In contrast to all other results presented in this paper, here we need to assume that TEST accepts arbitrary, not just nonnegative, cost coefficients. In particular, we set $c_{\max} := \max_{e \in E} |c_e|$.

[7] Otherwise one can transform the given instance into an equivalent one for which this is the case.

must be a solution $S' \in N(S)$ with $c(S') < c(S)$ and $S' \cap \{1, 2, \ldots, k\} = R$ such that $k+1 \in S'$. We leave R unchanged in the former case, and we set $R := R \cup \{k+1\}$ in the latter case.

Consequently, after n steps we have identified a set $S' = R \in N(S)$ such that $c(S') < c(S)$. □

Lemma 4.4 and Theorem 2.3 imply the following result.

COROLLARY 4.5. *An ε-local optimum of an instance $(\mathcal{F}, c)$ of a combinatorial optimization problem Π with neighborhood function N that is given via a* TEST *oracle of running time $\zeta(n, \log c_{\max})$ can be computed in time*

$$O(\xi(n) + \zeta(n, \log c_{\max}) n^2 \varepsilon^{-1} \min\{n \log n, \log K^0\}).$$

4.5. Bounded integer linear programming problems. Let us finally discuss some generalizations to the case of integer linear programs with bounded variables. In our discussions so far, we considered combinatorial optimization problems, which in fact are 0/1-integer linear programs, i.e., problems of the form $\min\{c\,x : x \in \mathcal{F}\}$ with $\mathcal{F} \subseteq \{0,1\}^n$. Interestingly, most of our results extend directly to the case of integer linear programs with bounded variables, which can be described as follows: $\min\{c\,x : x \in \mathcal{F}\}$ with $\mathcal{F} \subseteq \{0, 1, \ldots, u\}^n$ for some nonnegative integer u. In this context, a neighborhood assigns to each feasible solution $x \in \mathcal{F}$ a set of feasible points in $\{0, 1, \ldots, u\}^n$. If an initial feasible solution and an algorithm IMPROVE are available, the ε-local search algorithm can easily be modified to compute an ε-local optimum in this setting as well. In fact, by choosing the scaling parameter $q := \frac{K\varepsilon}{2n(1+\varepsilon)u}$, its number of iterations (and therefore the number of calls to IMPROVE) is $O(n\,\varepsilon^{-1} u \log K^0)$. Thus, if an initial feasible solution can be identified in polynomial time, if IMPROVE can be implemented in polynomial time, and if u is bounded by a polynomial in n and $\log c_{\max}$, then the ε-local search algorithm runs in polynomial time.

Acknowledgments. The authors are grateful to an anonymous referee for several insightful suggestions, which helped to improve the presentation of this paper. In particular, he or she brought the algorithm described in Figure 4.2 to their attention. They also thank Rafi Hassin for pointing them to the papers by Arkin and Hassin (1998) and Arkin et al. (2002) after reading an earlier version of this paper.

REFERENCES

R. K. AHUJA, Ö. ERGUN, AND A. P. PUNNEN (2002), *A survey of very large-scale neighborhood search techniques*, Discrete Appl. Math., 123, pp. 75–102.

E. M. ARKIN AND R. HASSIN (1998), *On local search for weighted k-set packing*, Math. Oper. Res., 23, pp. 640–648.

E. M. ARKIN, R. HASSIN, S. RUBINSTEIN, AND M. SVIRIDENKO (2002), *Approximations for maximum transportation problem with permutable supply vector and other capacitated star packing problems*, in Algorithm Theory—SWAT 2002, Proceedings of the 8th Scandinavian Workshop on Algorithm Theory, M. Penttonen and E. Meineche Schmidt, eds., Lecture Notes in Comput. Sci. 2368, Springer, Berlin, pp. 280–287.

G. AUSIELLO AND M. PROTASI (1995), *Local search, reducibility and approximability of NP-optimization problems*, Inform. Process. Lett., 54, pp. 73–79.

B. CHANDRA, H. KARLOFF, AND C. TOVEY (1999), *New results on the old k-opt algorithm for the traveling salesman problem*, SIAM J. Comput., 28, pp. 1998–2029.

W. J. COOK, W. H. CUNNINGHAM, W. R. PULLEYBLANK, AND A. SCHRIJVER (1998), *Combinatorial Optimization*, John Wiley & Sons, New York.

V. DEINEKO AND G. WOEGINGER (2000), *A study of exponential neighborhoods for the traveling salesman problem and the quadratic assignment problem*, Math. Program., 87, pp. 519–542.

S. T. FISCHER (1995), *A note on the complexity of local search problems*, Inform. Process. Lett., 53, pp. 69–75.

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

M. Grötschel and L. Lovász (1995), *Combinatorial optimization*, Chapter 28 in Handbook of Combinatorics, Vol. II, R. Graham, M. Grötschel, and L. Lovász, eds., North-Holland, Amsterdam, pp. 1541–1597.

G. Gutin, A. Yeo, and A. Zverovitch (2002), *Exponential neighborhoods and domination analysis for the TSP*, Chapter 6 in The Traveling Salesman Problem and Its Variations, G. Gutin and A. P. Punnen, eds., Kluwer Academic, Dordrecht, The Netherlands, pp. 223–256.

D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon (1989), *Optimization by simulated annealing: An experimental evaluation, part* I *(graph partitioning)*, Oper. Res., 37, pp. 865–892.

D. S. Johnson and L. A. McGeoch (1997), *The traveling salesman problem: A case study*, Chapter 8 in Local Search in Combinatorial Optimization, E. Aarts and J. K. Lenstra, eds., John Wiley & Sons, Chichester, UK, pp. 215–310.

D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis (1988), *How easy is local search?*, J. Comput. System Sci., 37, pp. 79–100.

B. W. Kernighan and S. Lin (1970), *An efficient heuristic procedure for partitioning graphs*, Bell System Tech. J., 49, pp. 291–307.

S. Khanna, R. Motwani, M. Sudan, and U. Vazirani (1998), *On syntactic versus computational views of approximability*, SIAM J. Comput., 28, pp. 164–191.

H. Klauck (1996), *On the hardness of global and local approximation*, in Algorithm Theory—SWAT '96, Proceedings of the 5th Scandinavian Workshop on Algorithm Theory, R. Karlsson, ed., Lecture Notes in Comput. Sci. 1097, Springer, Berlin, pp. 88–99.

B. Korte and J. Vygen (2002), *Combinatorial Optimization: Theory and Algorithms*, 2nd ed., Springer, Berlin.

M. W. Krentel (1989), *Structure in locally optimal solutions*, in Proceedings of the 30th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, pp. 216–221.

M. W. Krentel (1990), *On finding and verifying locally optimal solutions*, SIAM J. Comput., 19, pp. 742–749.

E. Lawler (1976), *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York.

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds. (1985), *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley & Sons, Chichester, UK.

S. Lin (1965), *Computer solutions of the traveling salesman problem*, Bell System Tech. J., 44, pp. 2245–2269.

C. H. Papadimitriou (1992), *The complexity of the Lin–Kernighan heuristic for the traveling salesman problem*, SIAM J. Comput., 21, pp. 450–465.

C. H. Papadimitriou and K. Steiglitz (1977), *On the complexity of local search for the traveling salesman problem*, SIAM J. Comput., 6, pp. 76–83.

C. H. Papadimitriou and K. Steiglitz (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

T. Radzik (1993), *Parametric flows, weighted means of cuts, and fractional combinatorial optimization*, in Complexity in Numerical Optimization, P. Pardalos, ed., World Scientific, River Edge, NJ, pp. 351–386.

S. Sahni and T. Gonzalez (1976), *P-complete approximation problems*, J. ACM, 23, pp. 555–565.

A. A. Schäffer and M. Yannakakis (1991), *Simple local search problems that are hard to solve*, SIAM J. Comput., 20, pp. 56–87.

A. S. Schulz and R. Weismantel (1999), *An oracle-polynomial time augmentation algorithm for integer programming*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, SIAM, Philadelphia, pp. 967–968.

A. S. Schulz and R. Weismantel (2002), *The complexity of generic primal algorithms for solving general integer programs*, Math. Oper. Res., 27, pp. 681–692.

A. S. Schulz, R. Weismantel, and G. M. Ziegler (1995), *0/1-integer programming: Optimization and augmentation are equivalent*, in Algorithms—ESA '95, Proceedings of the 3rd Annual European Symposium on Algorithms, P. Spirakis, ed., Lecture Notes in Comput. Sci. 979, Springer, Berlin, pp. 473–483.

A. S. Schulz, R. Weismantel, and G. M. Ziegler (1997), *An Optimization Problem Is Nine Problems*, talk presented by A. S. Schulz at the 16th International Symposium on Mathematical Programming, Lausanne, Switzerland.

M. Yannakakis (1997), *Computational complexity*, Chapter 2 in Local Search in Combinatorial Optimization, E. Aarts and J. K. Lenstra, eds., John Wiley & Sons, Chichester, UK, pp. 19–55.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1215–1222

A NOTE ON THE HOMOTOPY TYPE OF WAIT-FREE ATOMIC SNAPSHOT PROTOCOL COMPLEXES*

JOHN HAVLICEK†

Abstract. In the atomic snapshot system model, the processes of an asynchronous distributed system communicate by atomic write and atomic snapshot read operations on a shared memory consisting of single-writer multiple-reader registers. The processes may fail by crashing. It is shown that in this model, a wait-free full-information protocol complex is homotopy equivalent to the underlying input complex. A span in the sense of Herlihy and Shavit provides the homotopy equivalence. It follows that the protocol and input complexes are indistinguishable by ordinary homology or homotopy groups.

Key words. distributed computing, wait-free protocol, atomic snapshot, topology, homotopy, simplicial complex

AMS subject classifications. 68Q22, 55P10

DOI. 10.1137/S0097539798337224

1. Introduction.

Consider an asynchronous distributed system of $n+1$ processes that interact via shared objects, and assume that processes may fail by crashing. Coordination problems such as consensus and its generalizations, known as *decision tasks*, are fundamental to computation in such a system. A basic question of fault-tolerant computability in the system is whether a given decision task can be solved despite failure by various numbers of the constituent processes. A protocol that solves a decision task regardless of any pattern of failures by up to f processes is called *f-resilient*. A protocol that is n-resilient for a system of $n+1$ processes is said to be *wait-free*.

In this note, we restrict our attention to wait-free computation by deterministic protocols. By using Borowsky–Gafni simulation [BG2, B, LR], the question of f-resilient solvability of a decision task can in many cases be reduced to the question of wait-free solvability of essentially the same task in a smaller system. Thus, the theory of wait-free computation is of considerable significance.

It has been well established that insight into wait-free solvability of decision tasks is gained by introducing certain simplicial complexes and using standard techniques for studying their combinatorial and topological structure. The primer [HR] provides a good introduction to this area of research. Simplicial complexes are associated with the inputs and outputs of the task, and recent work [H] shows that it is also possible to define a simplicial complex representing the task relation. If a protocol is wait-free on an input complex, then there is an associated protocol complex, which is a geometric representation of the various possible executions of the protocol. The general structure of the protocol complex depends on the system model and is especially sensitive to the kinds of shared objects that are available. The success of the entire approach hinges on the fact that the topology of the simplicial complexes is computationally significant. Indeed, the combinatorial and topological relationships between the various complexes

*Received by the editors April 6, 1998; accepted for publication (in revised form) March 10, 2003; published electronically July 20, 2004. This work was supported by a Micro-electronics and Computer Development Fellowship granted by the Graduate School of The University of Texas at Austin.

http://www.siam.org/journals/sicomp/33-5/33722.html

†Motorola, Inc., 7700 W. Parmer Lane, Austin, TX 78729 (john.havlicek@motorola.com).

convey critical information about the solvability of the task and, if solvable, about the complexity of solution.

A typical system model in this line of research has a foundation of single-writer multiple-reader shared-memory registers. More sophisticated shared objects, such as (m, k)-consensus objects, may be added, significantly increasing the computational power of the system. For this note, however, we are interested in the underlying shared-memory register system. In addition to atomic writes, the system is assumed, without loss of generality [Af+], to provide atomic snapshot reads for accessing shared memory. This basic system will be referred to as the *atomic snapshot* (AS) system model. In order to maximize computational power in the model, it is generally assumed that protocols are structured to record full information. This means that each process locally accumulates a *view* (i.e., history) of the computation and maintains a copy of this view in shared memory. When a process reads, it appends a snapshot of the entire shared memory to its local view; and when a process writes, the value written is simply the current view of the process. It is not difficult to see that any other value that one might desire the process to write can be inferred from the view.

Several variations on the AS system model have been proposed. [BG1] introduces the *immediate snapshot* (IS) system model, whose executions are a strict subset of the AS executions. In the IS model, the individual write and read operations are fused into a single *WriteRead* operation. A set of processes may execute concurrent WriteRead operations, the effect being equivalent to an AS execution in which the processes of the set perform concurrent writes and then immediately perform concurrent snapshot reads. Another alternative is the *iterated immediate snapshot* (IIS) system model [B, BG3, HoS]. For the IIS system model, shared memory is divided into disjoint regions; one region is used for each immediate snapshot of the system. When a process executes its kth WriteRead operation, it writes a value into its single-writer portion of the kth memory region, and an immediate atomic snapshot of the kth memory region (only) is returned to the process. This snapshot value is then the value to be written by the process in the $(k+1)$th WriteRead operation.

It is known [BG3] that the AS, IS, and IIS system models are all computationally equivalent for wait-free solution of decision tasks. In other words, a decision task is wait-free solvable in one of the system models if and only if it is wait-free solvable in all three. However, the protocol complexes that arise from the different models are structurally distinct.

The IIS system model is the most restrictive. As a result, the associated protocol complexes have the simplest structure. If a protocol is normalized so that each process is directed to execute the same number of WriteRead operations (i.e., there is no early stopping), then the protocol complex is an iterated standard chromatic subdivision of the complex of input values [B, BG3, HS2]. Without normalization, the protocol complex is a nonuniform chromatic subdivision of the input complex [HoS]. In either case, an IIS protocol complex is homeomorphic to (i.e., of the same topological type as) the underlying input complex.

It has been claimed [BG3, p. 197] that protocol complexes in the IS system model are also chromatic subdivisions of the underlying input complexes, from which it follows that an IS protocol complex is also homeomorphic to the underlying input complex. No published proof of the claim is known to the author.

The purpose of this note is to clarify the structure of wait-free full-information protocol complexes in the AS system model. An AS protocol complex is not generally homeomorphic to the underlying input complex. Nevertheless, an AS protocol com-

plex is equivalent to the input complex in a weaker topological sense. In this note we show that a wait-free full-information AS protocol complex is homotopy equivalent to (i.e., of the same homotopy type as) the underlying input complex. In fact, a span in the sense of Herlihy and Shavit provides the homotopy equivalence.

Loosely speaking, homotopy equivalence is equivalence up to continuous deformation. It is a common notion in algebraic topology, and important standard functors, such as the ordinary homology and homotopy group functors, do not differentiate between spaces of the same homotopy type. Thus, from the point of view of ordinary homology or homotopy groups, a wait-free AS protocol complex is indistinguishable from the underlying input complex.

The fact that a wait-free AS protocol complex is homotopy equivalent to the underlying input complex gives an intuitive explanation for why the asynchronous computability theorem [HS2, HS3] works. Roughly speaking, the protocol complex holds computational information in topological form, and all of the topological structure of the protocol complex exists, up to homotopy equivalence, in the input complex. A span makes the translation between the two. The homotopy equivalence also clarifies the basis of the obstruction method introduced in [H] for detecting impossibility of solution of decision tasks. The obstruction method relies on the fact that a span admits a left homotopy inverse, and this note shows that the left homotopy inverse can be arranged as a two-sided homotopy inverse.

2. Preliminaries and notation. It is assumed that the reader is generally familiar with the use of simplicial complexes in the study of wait-free solvability of decision tasks. See [HR] for an introduction. Standard references for algebraic topology are [Mu] and [S].

2.1. Simplexes and complexes. If $\mathcal{A}$ is an abstract simplicial complex, then $|\mathcal{A}|$ denotes its polyhedron. Strictly speaking, $\mathcal{A}$ is a combinatorial object, while $|\mathcal{A}|$ is an associated topological space. If $f\colon \mathcal{A} \to \mathcal{B}$ is a simplicial map of simplicial complexes, then $|f|\colon |\mathcal{A}| \to |\mathcal{B}|$ denotes the associated piecewise-linear map between the polyhedra. $\mathcal{A}^{(k)}$ denotes the k-skeleton of $\mathcal{A}$, which is the subcomplex consisting of simplexes of $\mathcal{A}$ of dimension $\leq k$. If $\sigma(\mathcal{A})$ is a subdivision of $\mathcal{A}$ and S is a simplex of $\sigma(\mathcal{A})$, then carrier(S) denotes the smallest simplex of $\mathcal{A}$ whose polyhedron contains $|S|$. If A is a simplex, then ∂A denotes its boundary.

Fix a set L of labels. A *coloring* of simplex S is an injective function from the vertices of S to L. A *coloring* of simplicial complex $\mathcal{A}$ is a function χ from the vertices of $\mathcal{A}$ to L such that the restriction of χ to each simplex of $\mathcal{A}$ is a coloring of that simplex. A simplex or complex together with a coloring is called *chromatic*. If $\mathcal{A}$ and $\mathcal{B}$ are chromatic complexes, then a simplicial map $f\colon \mathcal{A} \to \mathcal{B}$ is called *chromatic*, provided $\chi(f(a)) = \chi(a)$ for each vertex a of $\mathcal{A}$. The set of labels used below is the set of identifiers of the processes of the distributed system.

An *input pair* is an ordered pair (p, v), where p is a process identifier and v is an input value. If $x = (p, v)$, then we write $\mathrm{id}(x) = p$ and $\mathrm{val}(x) = v$. A nonempty set $X = \{x_0, \ldots, x_r\}$ of input pairs forms an *input simplex*, provided that $\mathrm{id}(x_0), \ldots, \mathrm{id}(x_r)$ are distinct. In this case, X is an r-dimensional chromatic simplex with coloring $x \mapsto \mathrm{id}(x)$. We write $\mathrm{ids}(X) = \{\mathrm{id}(x_0), \ldots, \mathrm{id}(x_r)\}$. A set $\mathcal{K}$ of input simplexes is an *input complex*, provided that it satisfies the usual hereditary property of simplicial complexes: if $X \in \mathcal{K}$ and X' is a nonempty subset of X, then $X' \in \mathcal{K}$.

As a protocol executes, each process accumulates its local view of the computation. We assume that the view of a process begins with its input value. Therefore, the process copies its input value to shared memory in its first write operation. The

view continues, according to full information, with the sequence of snapshots of shared memory witnessed by the process. If a process completes the protocol, then its view at halting is its *final view* in the execution. A *protocol pair* is an ordered pair $z = (p, w)$, where p is a process identifier and w is the final view accumulated by p in some execution of the protocol. We write $\mathrm{id}(z) = p$ and $\mathrm{view}(z) = w$. Also, $\mathrm{inval}(z)$ denotes the input value for p in the execution. By assumption, $\mathrm{inval}(z)$ appears at the beginning of $\mathrm{view}(z)$. A nonempty set $Z = \{z_0, \dots, z_r\}$ of protocol pairs forms a *protocol simplex*, provided that there is a single execution e of the protocol such that for each $j = 0, \dots, r$ process $\mathrm{id}(z_j)$ accumulates $\mathrm{view}(z_j)$ as its final view in e. In this case, we say that e *certifies* Z. We write $\mathrm{ids}(Z) = \{\mathrm{id}(z_0), \dots, \mathrm{id}(z_r)\}$.

If z is a protocol pair, then there may exist input pairs other than $(\mathrm{id}(z), \mathrm{inval}(z))$ that can be inferred from z. This is because the snapshots in $\mathrm{view}(z)$ may witness the input values copied by other processes into shared memory. By $\Xi(z)$ we mean the input simplex consisting of all input pairs that are determined in this fashion from z. If Z is a protocol simplex, then we let

$$\Xi(Z) = \bigcup_{z \in Z} \Xi(z).$$

There is a chromatic simplicial map $\psi\colon Z \to \Xi(Z)$ defined by $z \mapsto (\mathrm{id}(z), \mathrm{inval}(z))$.

Consider a full-information AS protocol that is wait-free on the input simplex X. By $\mathcal{P}(X)$ we mean the *protocol complex* of all protocol simplexes that can be certified by executions with X as input simplex:

$$\mathcal{P}(X) = \{Z : \Xi(Z) \subseteq X\}.$$

Note in particular that $Z \in \mathcal{P}(\Xi(Z))$. If $\mathcal{K}$ is an input complex on which the protocol is wait-free, then $\mathcal{P}(\mathcal{K})$ is the union of the complexes $\mathcal{P}(X)$ as X ranges over the various input simplexes of $\mathcal{K}$. In this case, there is a chromatic simplicial map $\psi\colon \mathcal{P}(\mathcal{K}) \to \mathcal{K}$ defined as above.

2.2. Homotopy. By *space* we mean a topological space embedded in a finite-dimensional Euclidean space. By *map* we mean a continuous function. Let X and Y be spaces, and let $f, g\colon X \to Y$ be maps. A *homotopy* from f to g is a map

$$F\colon X \times [0,1] \to Y$$

that satisfies

$$F(-,0) = f \quad \text{and} \quad F(-,1) = g.$$

In other words, when suitably identifying the ends of the cylinder $X \times [0,1]$ with X, the restriction of F to one end is the map f, and the restriction of F to the other end is the map g. Furthermore, the restriction of F to the cross section $X \times \{t\}$ is a map $F(-,t)\colon X \to Y$. Continuity of F on the cylinder ensures that as t moves from 0 to 1, $F(-,t)$ "deforms continuously" from f to g. If there is a homotopy from f to g, then f and g are said to be *homotopic*.

Example 2.2.1. Let X and Y be nonempty spaces with Y convex, and let $f, g\colon X \to Y$ be maps. Define $F\colon X \times [0,1] \to Y$ by

$$F(x,t) = (1-t)f(x) + tg(x).$$

F is into Y because $(1-t)f(x) + tg(x)$ parameterizes the line segment $[f(x), g(x)]$, which lies in Y by convexity. The continuity of f and g ensures that F is continuous.

Thus, f and g are homotopic. F is often referred to as a *straight-line homotopy* between f and g. It may be possible to form a straight-line homotopy even if Y is not globally convex. If for every $x \in X$ both $f(x)$ and $g(x)$ lie in a convex neighborhood of Y, then F is a straight-line homotopy between f and g. □

Let $f\colon X \to Y$ and $g\colon Y \to X$ be maps. If $f \circ g$ is homotopic to the identity map of Y, then g is called a *right homotopy inverse* for f, and if $g \circ f$ is homotopic to the identity map of X, then g is called a *left homotopy inverse* for f. If g is both a left and a right homotopy inverse for f, then the same is true of f for g. In this case, (1) f and g are called *homotopy inverses* of one another, (2) each of f and g is said to be a *homotopy equivalence*, and (3) the spaces X and Y are said to be *homotopy equivalent*.

A space X is *contractible* if it is homotopy equivalent to a one-point space. Any nonempty convex space is contractible, and thus the polyhedron of any simplex is contractible. A map $f\colon X \to Y$ is called *null-homotopic* if f is homotopic to a constant map. Any map from a nonempty space to a contractible space is null-homotopic. The following extension property is well known.

PROPOSITION 2.2.2. *Let X be homeomorphic to a closed disk, let ∂X be the boundary sphere of X, and let Y be a nonempty space. A map $f\colon \partial X \to Y$ can be extended to a map $X \to Y$ if and only if f is null-homotopic.* □

Note that if A is a simplex, then Proposition 2.2.2 can be applied with $X = |A|$ and $\partial X = |\partial A|$. Since $|\partial A|$ has a collar neighborhood in $|A|$, one has the following special case of the simplicial approximation theorem.

PROPOSITION 2.2.3. *Let A be a simplex, let $\sigma_0(\partial A)$ be a subdivision of its boundary, and let $\mathcal{B}$ be a simplicial complex. Suppose that there is a simplicial map $\varphi_0\colon \sigma_0(\partial A) \to \mathcal{B}$. If $|\varphi_0|$ is null-homotopic, then there is a subdivision $\sigma(A)$ and there is a simplicial map $\varphi\colon \sigma(A) \to \mathcal{B}$ such that $\sigma(\partial A) = \sigma_0(\partial A)$ and such that the restriction of φ to $\sigma(\partial A)$ equals φ_0.* □

The next result is fundamental and lies at the heart of the asynchronous computability theorem of Herlihy and Shavit. It can be proved by generalizing the critical state technique from the seminal paper [FLP] of Fischer, Lynch, and Paterson.

THEOREM 2.2.4 (contractibility theorem [HS1]). *Assume that an AS protocol records full information, and let X be an input simplex on which the protocol is wait-free. Then $|\mathcal{P}(X)|$ is contractible.* □

In [HS3], Herlihy and Shavit actually prove that $|\mathcal{P}(X)|$ has trivial fundamental group and trivial reduced homology groups in all dimensions. The theorem of Whitehead [S, p. 399] and standard homotopy theory [S, pp. 405–406] imply that these conditions are equivalent to the contractibility of $|\mathcal{P}(X)|$.

3. Wait-free AS protocol complexes. Consider a full-information AS protocol that is wait-free on the input complex $\mathcal{K}$. We show in Theorem 3.2.2 that the map

$$|\psi|\colon |\mathcal{P}(\mathcal{K})| \to |\mathcal{K}|$$

is a homotopy equivalence and that if $\varphi\colon \sigma(\mathcal{K}) \to \mathcal{P}(\mathcal{K})$ is a span in the sense of Herlihy and Shavit, then $|\varphi|$ is a homotopy inverse for $|\psi|$. The existence of spans is proved in [HS3], but the condition that φ be chromatic is not needed in order to prove the homotopy equivalence. In [HS3], most of the technical work to prove the existence of spans is concerned with arranging the chromatic condition. To avoid assuming these arguments, we review in subsection 3.1 the simple proof of the existence of

"nonchromatic" spans that appears in [HS3]. This argument also serves as a model for the proof of Theorem 3.2.2.

3.1. Nonchromatic spans. A *nonchromatic span* (for the protocol on $\mathcal{K}$) is a simplicial map φ from a subdivision $\sigma(\mathcal{K})$ of $\mathcal{K}$ to $\mathcal{P}(\mathcal{K})$ such that for any simplex S of $\sigma(\mathcal{K})$, $\varphi(S) \in \mathcal{P}(\text{carrier}(S))$. Note that if $\varphi\colon \sigma(\mathcal{K}) \to \mathcal{P}(\mathcal{K})$ is a nonchromatic span and if $\mathcal{L}$ is a subcomplex of $\mathcal{K}$, then the restriction of φ to $\sigma(\mathcal{L})$ is a nonchromatic span for the protocol on $\mathcal{L}$.

THEOREM 3.1.1 (see [HS3]). *Assume that an AS protocol records full information, and let $\mathcal{K}$ be an input complex on which the protocol is wait-free. Then there exists a nonchromatic span for the protocol on $\mathcal{K}$.*

Proof. We construct the subdivision $\sigma(\mathcal{K})$ and the map φ inductively on the skeleta of $\mathcal{K}$. The 0-skeleton of $\mathcal{K}$ admits no subdivision. Let x be a vertex of $\mathcal{K}$, and let $p = \text{id}(x)$. Since the protocol is wait-free, there is a finite execution e in which p starts with input value $\text{val}(x)$ and only p has events. Let $z = (p, w)$, where w is the final view of p in e. Then $\mathcal{P}(\{x\})$ has only the single vertex z. We must, therefore, let $\varphi(x) = z$. This defines φ as a nonchromatic span for the protocol on $\mathcal{K}^{(0)}$.

Assume now that $k > 0$, that the subdivision $\sigma(\mathcal{K}^{(k-1)})$ has been defined, and that φ has been defined as a nonchromatic span for the protocol on $\mathcal{K}^{(k-1)}$. Let X be a k-simplex of $\mathcal{K}$. The boundary ∂X is contained in $\mathcal{K}^{(k-1)}$, so φ gives a simplicial map $\sigma(\partial X) \to \mathcal{P}(X)$. According to the contractibility theorem, $|\mathcal{P}(X)|$ is contractible, and so the restriction of $|\varphi|$ to $|\partial X|$ is null-homotopic. By Proposition 2.2.3, the subdivision $\sigma(\partial X)$ can be extended to a subdivision $\sigma(X)$, and φ can be extended to a simplicial map $\sigma(X) \to \mathcal{P}(X)$. In this way, we define the subdivision $\sigma(\mathcal{K}^{(k)})$ and the map $\varphi\colon \sigma(\mathcal{K}^{(k)}) \to \mathcal{P}(\mathcal{K}^{(k)})$. Suppose that S is simplex of $\sigma(\mathcal{K}^{(k)})$. If S lies in $\sigma(\mathcal{K}^{(k-1)})$, then $\varphi(S) \in \mathcal{P}(\text{carrier}(S))$ by the inductive hypothesis. Otherwise, $\text{carrier}(S)$ is a k-simplex of $\mathcal{K}$, say X. From our construction, we see that φ maps S into $\mathcal{P}(X) = \mathcal{P}(\text{carrier}(S))$. This verifies that φ is a nonchromatic span for the protocol on $\mathcal{K}^{(k)}$ and completes the inductive step. □

3.2. The homotopy type of wait-free AS protocol complexes. In this subsection, we show that $|\mathcal{K}|$ and $|\mathcal{P}(\mathcal{K})|$ are homotopy equivalent spaces. According to Theorem 3.1.1, there is a nonchromatic span $\varphi\colon \sigma(\mathcal{K}) \to \mathcal{P}(\mathcal{K})$. We first show that $|\varphi|$ is a right homotopy inverse for $|\psi|$.

PROPOSITION 3.2.1. *$|\psi| \circ |\varphi|$ is homotopic to the identity map of $|\mathcal{K}|$.*

Proof. Without loss of generality, $|\mathcal{K}| = |\sigma(\mathcal{K})|$. For any simplex S in $\sigma(\mathcal{K})$,

$$\psi \circ \varphi(S) \subseteq \text{carrier}(S).$$

Since $|S| \subseteq |\text{carrier}(S)|$ and since $|\text{carrier}(S)|$ is convex, we can form a straight-line homotopy between $|\psi| \circ |\varphi|$ and the identity map of $|\mathcal{K}|$. □

THEOREM 3.2.2. *Assume that an AS protocol records full information. Let $\mathcal{K}$ be an input complex on which the protocol is wait-free, and let $\varphi\colon \sigma(\mathcal{K}) \to \mathcal{P}(\mathcal{K})$ be a nonchromatic span. Then $|\varphi|$ and $|\psi|$ are homotopy inverses, and so $|\mathcal{P}(\mathcal{K})|$ and $|\mathcal{K}|$ are homotopy equivalent.*

Proof. From Proposition 3.2.1 we have that $|\psi| \circ |\varphi|$ is homotopic to the identity map of $|\mathcal{K}|$. We now construct a homotopy

$$F\colon |\mathcal{P}(\mathcal{K})| \times [0, 1] \to |\mathcal{P}(\mathcal{K})|$$

from $|\varphi| \circ |\psi|$ to the identity map of $|\mathcal{P}(\mathcal{K})|$ by induction on the skeleta of $\mathcal{P}(\mathcal{K})$. F will be arranged so that for all $Z \in \mathcal{P}(\mathcal{K})$,

$$(*) \qquad F(|Z| \times [0,1]) \subseteq |\mathcal{P}(\Xi(Z))| \,.$$

Let z be a vertex of $\mathcal{P}(\mathcal{K})$. Then z is a vertex of $\mathcal{P}(\Xi(z))$. Also $\psi(z)$ is a vertex of $\Xi(z)$, and hence $\varphi \circ \psi(z)$ is a vertex of $\mathcal{P}(\Xi(z))$. Since $|\mathcal{P}(\Xi(z))|$ is contractible, there is a path in $|\mathcal{P}(\Xi(z))|$ from $|\varphi| \circ |\psi|(|z|)$ to $|z|$. This path defines F on $|z| \times [0,1]$. Combining these paths defines F on $|\mathcal{P}(\mathcal{K})^{(0)}| \times [0,1]$ in a way that satisfies $(*)$ for all vertices of $\mathcal{P}(\mathcal{K})$.

Now let $k > 0$, and assume that F has been defined on $|\mathcal{P}(\mathcal{K})^{(k-1)}| \times [0,1]$ and satisfies $(*)$ for all simplexes in $\mathcal{P}(\mathcal{K})^{(k-1)}$. Let Z be a k-simplex of $\mathcal{P}(\mathcal{K})$. Then $Z \in \mathcal{P}(\Xi(Z))$, and hence $|Z| \subseteq |\mathcal{P}(\Xi(Z))|$. Also $\psi(Z) \subseteq \Xi(Z)$, and hence $|\varphi| \circ |\psi|(|Z|) \subseteq |\mathcal{P}(\Xi(Z))|$. Let

$$\Sigma \;=\; |Z| \times \{0\} \;\cup\; |\partial Z| \times [0,1] \;\cup\; |Z| \times \{1\}.$$

Notice that Σ is the boundary of the closed $(k+1)$-disk $|Z| \times [0,1]$, and hence Σ is a k-sphere. By the inductive hypothesis, F is already defined on $|\partial Z| \times [0,1]$ and satisfies $F(|\partial Z| \times [0,1]) \subseteq |\mathcal{P}(\Xi(Z))|$. For $x \in |Z|$, let $F(x,0) = |\varphi| \circ |\psi|(x)$ and $F(x,1) = x$. These definitions of F cohere to give a map $\Sigma \to |\mathcal{P}(\Xi(Z))|$. Since $|\mathcal{P}(\Xi(Z))|$ is contractible, Proposition 2.2.2 implies that F can be extended to a map of the disk $|Z| \times [0,1] \to |\mathcal{P}(\Xi(Z))|$. Combining these extensions for the various k-simplexes of $\mathcal{P}(\mathcal{K})$ gives the extension of F to $|\mathcal{P}(\mathcal{K})^{(k)}| \times [0,1]$. By construction, $(*)$ holds for all simplexes in $\mathcal{P}(\mathcal{K})^{(k)}$. □

REFERENCES

[Af+] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. ACM, 40 (1993), pp. 873–890.

[B] E. Borowsky, *Capturing the Power of Resiliency and Set Consensus in Distributed Systems*, Ph.D. Thesis, Computer Science Department, UCLA, Los Angeles, CA, 1995.

[BG1] E. Borowsky and E. Gafni, *Immediate atomic snapshots and fast renaming*, in Proceedings of the 12th ACM Symposium on Principles of Distributed Computing (PODC), Ithaca, NY, 1993, pp. 41–51.

[BG2] E. Borowsky and E. Gafni, *Generalized FLP impossibility result for t-resilient asynchronous computations*, in Proceedings of the 25th ACM Symposium on Theory of Computing (STOC), San Diego, CA, 1993, pp. 91–100.

[BG3] E. Borowsky and E. Gafni, *A simple algorithmically reasoned characterization of wait-free computations*, in Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC), Santa Barbara, CA, 1997, pp. 189–198.

[FLP] M. Fischer, N. Lynch, and M. Paterson, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.

[H] J. Havlicek, *Computable obstructions to wait-free computability*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS), Miami Beach, FL, 1997, pp. 80–89.

[HR] M. Herlihy and S. Rajsbaum, *A primer on algebraic topology and distributed computing*, in Computer Science Today, Jan van Leeuwen, ed., Lecture Notes in Comput. Sci. 1000, Springer, New York, 1996, pp. 203–217.

[HS1] M. Herlihy and N. Shavit, *The asynchronous computability theorem for t-resilient tasks*, in Proceedings of the 25th ACM Symposium on Theory of Computing (STOC), San Diego, CA, 1993, pp. 111–120.

[HS2] M. Herlihy and N. Shavit, *A simple constructive computability theorem for wait-free computation*, in Proceedings of the 26th ACM Symposium on Theory of Computing (STOC), Montreal, 1994, pp. 101–110.

[HS3] M. Herlihy and N. Shavit, *The Topological Structure of Asynchronous Computability*, Technical Report TR CS-96-03, Brown University, Providence, RI, 1996.

[HoS] G. Hoest and N. Shavit, *Towards a topological characterization of asynchronous complexity*, in Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC), Santa Barbara, CA, 1997, pp. 199–208.

[LR] N. Lynch and S. Rajsbaum, *On the Borowsky–Gafni simulation algorithm*, a brief announcement in Proceedings of the 15th ACM Symposium on Principles of Distributed Computing (PODC), Philadelphia, 1996, p. 57.

[Mu] J. Munkres, *Elements of Algebraic Topology*, Addison-Wesley, Reading, MA, 1984.

[S] E. Spanier, *Algebraic Topology*, McGraw–Hill, New York, 1966; Springer, 1982.

SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1223–1260

AVERAGE-CASE PERFORMANCE OF THE APRIORI ALGORITHM*

PAUL W. PURDOM†, DIRK VAN GUCHT†, AND DENNIS P. GROTH‡

Abstract. The failure rate of the Apriori Algorithm is studied analytically for the case of random shoppers. The time needed by the Apriori Algorithm is determined by the number of item sets that are output (successes: item sets that occur in at least k baskets) and the number of item sets that are counted but not output (failures: item sets where all subsets of the item set occur in at least k baskets but the full set occurs in less than k baskets). The number of successes is a property of the data; no algorithm that is required to output each success can avoid doing work associated with the successes. The number of failures is a property of both the algorithm and the data.

We find that under a wide range of conditions the performance of the Apriori Algorithm is almost as bad as is permitted under sophisticated worst-case analyses. In particular, there is usually a *bad level* with two properties: (1) it is the level where nearly all of the work is done, and (2) nearly all item sets counted are failures. Let l be the level with the most successes, and let the number of successes on level l be approximately $\binom{m}{l}$ for some m. Then, typically, the Apriori Algorithm has total output proportional to approximately $\binom{m}{l}$ and total work proportional to approximately $\binom{m}{l+1}$. In addition m is usually much larger than l, so the ratio of work to output is proportional to approximately $m/(l+1)$.

The analytical results for random shoppers are compared against measurements for three data sets. These data sets are more like the usual applications of the algorithm. In particular, the buying patterns of the various shoppers are highly correlated. For most thresholds, these data sets also have a bad level. Thus, under most conditions nearly all of the work done by the Apriori Algorithm consists in counting item sets that fail.

Key words. data mining, algorithm analysis, Apriori Algorithm

AMS subject classification. 68W40

DOI. 10.1137/S0097539703422881

1. Introduction.

The Apriori Algorithm [2, 3, 6, 16] solves the frequent item sets problem, which is at the core of various algorithms for data mining problems. The best known such problem is the problem of finding the *association rules* that hold in a basket-items relation [2, 3, 16, 20]. Other data mining problems based on the Apriori Algorithm are discussed in [6, 14, 16, 18, 21, 22].

The Apriori Algorithm analyzes a data set of baskets, each containing a set of items, to determine which combination of items occur together frequently. Consider a store with $|I|$ items where b shoppers each have a single basket. Each shopper selects a set of items for their basket. The input to the Apriori Algorithm is a list giving the contents of each basket. For a fixed threshold k, the algorithm outputs a list of those sets of items that are *frequent*; that is, they are contained in at least k of the b baskets.

The Apriori Algorithm is a level-wise algorithm. It considers sets of items in order of their size: first sets of size one are tested to see whether they are contained in k baskets, then sets of size two, etc. On each level, the algorithm knows the result

*Received by the editors February 14, 2003; accepted for publication (in revised form) January 26, 2004; published electronically July 29, 2004. The work of the second and third authors was supported by NSF grant IIS-0082407.

http://www.siam.org/journals/sicomp/33-5/42288.html

†Computer Science Department, Indiana University, Bloomington, IN 47405-4101 (pwp@cs.indiana.edu, vgucht@cs.indiana.edu).

‡School of Informatics, Indiana University, Bloomington, IN 47405-4101 (dgroth@indiana.edu).

of the previous level. The key idea of the Apriori Algorithm is that, on level l, a set is tested if and only if all its subsets of size $l-1$ are frequent. An item set satisfying this property is called a *candidate* (at level l). It is a *success* (at level l) if it is contained in at least k baskets, and it is a *failure* otherwise.

This paper considers only the Apriori Algorithm. Many of its performance characteristics come from the fact that it outputs every frequent item set. There are several interesting algorithms for the frequent item set problem that output only the maximal frequent item sets, since every subset of a frequent item set is also frequent [1, 4, 12]. Also, there is an algorithm complementary to the Apriori Algorithm that finds the infrequent item sets, starting with the set of all items and working downward in set size.

This paper considers the expected amount of work that the Apriori Algorithm does when the shoppers shop at random. Specifically, the probability that a shopper buys an item is p, independent of all other shoppers and all other items. This is the same probability model that has previously been used to estimate the probability that a set is frequent [3, 17, 21]. The main advantage of using a parameterized probability model is that we can study the performance of the algorithm under a wide range of conditions. While the later sections contain many mathematical details, the main conclusions are fairly simple. For most values of the parameters, random data result in a high success rate (essentially 1) for small values of l, with a sudden switch to a low success rate (essentially 0) for larger values of l. When l is below the level of the switch, the number of candidates is just below $\binom{|I|}{l}$ and almost all of the candidates are successes. When l is above the level of the switch, almost all candidates are failures, and the number of candidates decreases rapidly. The level for the switch depends on the parameters. When l is much less than $|I|$, nearly all the work of the algorithm is done testing candidates from the level with the most successes. The algorithm has a *bad level* where (1) most of the work is done and (2) there are a lot of candidates, but few successes. For a few thresholds, two levels in a row are bad.

The main disadvantage of using a simple probability model is that most applications of the Apriori Algorithm involve data sets with complex correlations, and such data sets are difficult to model mathematically. Therefore, we also measured the success, candidate, and failure rates of the performance of the Apriori Algorithm on three data sets: (1) synthetic data generated by the generator from the IBM Quest Research Group [13]; (2) U.S. Census data, using Public Use Microdata Samples (PUMS) (the same sample that was used by [5, 6] and processed in the same way); and (3) a Web data set [24]. In one important way, the results with these data sets were similar to that of random shoppers. For most thresholds, the experimental data show the existence of a single bad level. This bad level came just after the level with the most successes, it had a low success rate, and it consumed most of the work that the algorithm did. On the other hand, with the experimental data there are also many failures below the bad level, so that the number of candidates is much less than $\binom{|I|}{l}$.

2. The Apriori Algorithm. Let J_l be a set of $l \geq 1$ items where the items are selected from the set I. For a given J_l, define J_l^{-h} to be the set obtained from J_l by omitting element h, where h is an element of J_l. The key idea of the Apriori Algorithm is that the set J_l cannot possibly have k occurrences unless, for each h in J_l, the set J_l^{-h} has k occurrences. Since the algorithm considers possible sets in order of their size, it has already gathered the information about all the sets of size $l-1$ before it considers sets of size l.

APRIORI ALGORITHM.
Step 1. For l from 1 to $|I|$ do
Step 2. For each set J_l such that for each $h \in J_l$ the set J_l^{-h} occurs in at least k baskets do
Step 3. Examine the data to determine whether the set J_l occurs in at least k baskets. Remember those cases where the answer is "yes."

Step 2 of the algorithm generates each set J_l such that J_l^{-h} occurs at least k times (for each h in J_l). The sets that are generated are called *candidates*. For those sets that are candidates, Step 3 examines the data (basket contents) to determine which set of items occurs in at least k baskets. This counting and comparing with the threshold is called the *frequency test*. A candidate that passes the frequency test is called a *success* (or a *frequent item set*). A set that is subjected to the frequency test but fails is called a *failure*. For algorithms that verify each success by doing the frequency test, the main place for improvement is to reduce the number of failures.

For typical data sets, a careful implementation of the Apriori Algorithm will have it spending most of its time accessing the data base (counting the occurrences of the various candidates). The implementation should exit the loop in Step 1 early if there are no "yes" answers for some value of l. On level l, it should consider only those sets that are formed from sets that passed the frequency test on level $l-1$. In addition, no set of size l should be generated more than once. The sets can be generated by assigning an order to the items and extending each set S on level $l-1$ only with items that are greater than the largest item in S. *Assuming unit time for hash table look-ups* (for looking up various subsets of the extended S), the algorithm can do the work for a single candidate set on level l in time bounded by a constant times $l+1$. See [2] for further discussion of the techniques used in good implementations.

Although the Apriori Algorithm uses explicit counting to verify that an item set is frequent, this is not always logically necessary. Thus, if item A occurs in b_A baskets and item B occurs in b_B baskets, then A and B must occur together in at least $b_A + b_B - b$ baskets [11]. Similar ideas are explored in [7].

3. Best and worst cases. We use the number of candidates as a proxy for the amount of computing that the Apriori Algorithm does. Let N_S be the number of item sets that are successes. Let N_F be the number of failures (candidates that are not successes). The total work is proportional to $N_S + N_F$. The term N_S represents work that must be done by any algorithm that outputs every frequent item set, and it is a property of the data, not of the algorithm. On the other hand, N_F represents work that we might hope to avoid.

The worst-case output is exponentially larger than the input (every set might be frequent), so the worst-case time needed can be exponential in the size of the input. A closely related problem that is NP-complete is to determine whether or not there are any sets of size l that occur k times; this is NP-complete because the balanced complete bipartite subgraph problem [9] reduces to it. The appendix has the details of proofs for this and many other statements.

For algorithms that must output every frequent item set, the ratio $N_S/(N_S+N_F)$ is the natural measure of algorithm efficiency. When every item set is frequent, N_S is $2^{|I|}$ and $N_F = 0$, so the algorithm is efficient. The worst ratio of work to output occurs when no (nonempty) item set is frequent, in this case $N_S = 1$, $N_F = |I|$. (Most versions of the algorithm avoid outputting the empty set, leading to values of N_S one less than above.)

Intermediate amounts of output are more interesting. In such cases, one can have

a lot of output and also a high failure rate. In this regard, the worst-case bounds of Geerts, Goethals, and Van den Bussche [10] are interesting. For their Theorem 1, they write the number of successes on level l as a binomial coefficient with l on the bottom in which the top index is as large as possible subject to the binomial coefficient being no larger than the number of success. If the difference between the number of successes and the binomial coefficient is positive, then they repeat the process with the difference. Thus, they write the number of successes as a sum of binomial terms. They show that the result of increasing each bottom index to $l+1$ gives an upper bound on the number of candidates for level $l+1$. This bound is exact for some data sets, including one that results in no failures on level l. Experiments show that in many cases this upper bound on candidates is close to the exact value for level $l+1$. The same idea can be used to predict the number of successes on later levels, but it often works less well for those levels.

The following argument also limits the number of candidates on level $l+1$. Consider the graph where each candidate on level $l+1$ (each J_{l+1}) is connected to each of its associated frequent items sets (J_{l+1}^{-h}). Thus, there are $[N_S(l+1)+N_F(l+1)](l+1)$ arcs. Each of the $N_S(l)$ frequent item sets on level l is connected to at most $|I|-l$ candidates on level $l+1$. Indeed, if we use $|I_l|$ to be the number of items that occur among the frequent item sets of level l, there are at most $|I_l|-l$ connections from a single level l frequent item set. Thus, we have

$$[N_S(l+1)+N_F(l+1)](l+1) \le N_S(l)(|I_l|-l), \tag{1}$$

$$[N_S(l+1)+N_F(l+1)] \le \left(\frac{|I_l|-l}{l+1}\right) N_S(l). \tag{2}$$

This gives

$$\sum_{0\le l\le |I_l|-1} \left(\frac{|I_l|-l}{l+1}\right) N_S(l) \tag{3}$$

as an upper bound on the amount of work. In many cases, $N_S(l)$ increases rapidly for small l and then suddenly drops rapidly to zero. In such cases, the largest term in this sum is a good approximation to its total value.

We see that the Apriori Algorithm is rather efficient for algorithms that output every frequent item set. The total work is never more than a constant plus a factor $|I|$ times larger than N_S, the least amount of work that could possibly be done by any algorithm in the class. When most of the work is concentrated on level $l+1$, the amount of work is better than this product by a factor of $l+1$.

4. Average case. We now start an exact computation of the average-case performance of the Apriori Algorithm for the case when the baskets are filled at random. That is, each basket j has item i with probability p independent of what happens for other baskets and other items. We eventually show that for most values of the parameters, the average performance is not significantly better than that suggested by the worst-case analysis (equation (3)).

Let S_l be the probability that the set consisting of items 1 to l is a frequent item set, and let F_l be the probability that the same set is a candidate but fails to be a frequent item set. Since each basket is filled randomly, any other set of l items has

the same probability of success and failure. The expected number of successes is

$$\sum_{1\le l\le |I|} \binom{|I|}{l} S_l, \tag{4}$$

and the expected number of failures is

$$\sum_{1\le l\le |I|} \binom{|I|}{l} F_l. \tag{5}$$

The number of item sets for which the basket data is examined is

$$\sum_{1\le l\le |I|} \binom{|I|}{l} (S_l + F_l). \tag{6}$$

Under the above assumptions, the running time is bounded by a constant times

$$\sum_{1\le l\le |I|} (l+1)\binom{|I|}{l} (S_l + F_l). \tag{7}$$

Define the following conditions with respect to a single basket:

- M_0: the basket has all the items 1 to l, and
- M_h $(1 \le h \le l)$: the basket has all items from 1 to l except that it does *not* have item h.

These conditions are disjoint; each basket obeys at most one of the conditions M_h, $0 \le h \le l$.

The probability that a randomly filled basket obeys condition M_0 is

$$P(l) = p^l \tag{8}$$

(when $l \le |I|$). The probability that a randomly filled basket obeys condition M_h (for any h in the range 1 to l) is

$$Q(l) = p^{l-1}(1-p). \tag{9}$$

Note that

$$P(l-1) = P(l) + Q(l). \tag{10}$$

The probability that at least k baskets obey condition M_0 is

$$S_l = \sum_{j\ge k} \binom{b}{j} [P(l)]^j [1-P(l)]^{b-j} = 1 - \sum_{j<k} \binom{b}{j} [P(l)]^j [1-P(l)]^{b-j}. \tag{11}$$

The probability that j_0 baskets obey condition M_0, j_1 baskets obey condition M_1, $\ldots$, j_l baskets obey condition M_l, and the remaining $b - j_0 - \cdots - j_l$ baskets do not obey any of the conditions is

$$\binom{b}{j_0, \ldots, j_l, b - j_0 - \cdots - j_l} [P(l)]^{j_0} [Q(l)]^{j_1+\cdots+j_l} [1 - P(l) - lQ(l)]^{b-j_0-\cdots-j_l}, \tag{12}$$

where the multinomial coefficient is the number of ways to arrange b distinct baskets into $l+1$ sets, where set 0 has j_0 baskets, ..., set l has j_l baskets, and $b-j_0-\cdots-j_l$ baskets are not in any of the $l+1$ sets.

The item set $\{1,\ldots,l\}$ is a candidate if and only if for each h in the range $1 \le h \le l$ we have the number of baskets satisfying condition M_0 plus the number of baskets satisfying condition M_h totaling at least k. Thus, item set $\{1,\ldots,l\}$ is a candidate in just those cases where the conditions

$$j_0 + j_1 \ge k,\ j_0 + j_2 \ge k,\ \ldots,\ j_0 + j_l \ge k \tag{13}$$

are all true. Thus, the probability that the set $\{1,\ldots,l\}$ is a candidate is the above probability (12) summed over those cases that satisfy the conditions (13),

$$\begin{aligned} C_l = \sum_{\substack{j_0 \\ j_1\ge k-j_0 \\ j_2\ge k-j_0 \\ \cdots \\ j_l\ge k-j_0}} & \binom{b}{j_0,\ldots,j_l,b-j_0-\cdots-j_l} \\ & \times [P(l)]^{j_0}[Q(l)]^{j_1+\cdots+j_l}[1-P(l)-lQ(l)]^{b-j_0-\cdots-j_l}. \end{aligned} \tag{14}$$

Since the set $\{1,\ldots,l\}$ either does or does not occur in at least k baskets, the probability that the item set $\{1,\ldots,l\}$ is a candidate but not a frequent item set is

$$\begin{aligned} F_l = C_l - S_l = \sum_{\substack{j_0<k \\ j_1\ge k-j_0 \\ j_2\ge k-j_0 \\ \cdots \\ j_l\ge k-j_0}} & \binom{b}{j_0,\ldots,j_l,b-j_0-\cdots-j_l} \\ & \times [P(l)]^{j_0}[Q(l)]^{j_1+\cdots+j_l}[1-P(l)-lQ(l)]^{b-j_0-\cdots-j_l}. \end{aligned} \tag{15}$$

4.1. Efficient computation of F_l. The number of arithmetic operations needed to compute S for fixed b, l, and p (using the right part of (11)) is $O(k)$. Furthermore, the number of operations for fixed l and p and for all k is only $O(b)$.

The number of operations needed to compute F by direct application of (15) is $O(kb^l)$. However, using the recurrence equations below, F can be computed in time that is independent of k and polynomial in b and l.

Write (15) as

$$F_l = \sum_{j_0<k} \binom{b}{j_0} [P(l)]^{j_0} R_{k-j_0}(b-j_0,l,l,l), \tag{16}$$

where

$$\begin{aligned} R_k(b,l,m,n) = \sum_{\substack{j_1\ge k \\ j_2\ge k \\ \cdots \\ j_l\ge k}} & \binom{b}{j_1,\ldots,j_l,b-j_1-\cdots-j_l} \\ & \times [Q(m)]^{j_1+\cdots+j_l}[1-P(m)-nQ(m)]^{b-j_1-\cdots-j_l}. \end{aligned} \tag{17}$$

By considering the sum over j_l (represented by j in the sum below) separately, we have

$$R_k(b,l,m,n) = \sum_{j\ge k} \binom{b}{j} [Q(m)]^j R_k(b-j,l-1,m,n), \tag{18}$$

with boundary condition

$$R_k(b,0,m,n) = [1 - P(m) - nQ(m)]^b. \tag{19}$$

With these equations, a particular $R_k(b,l,m,n)$ can be computed from the various $R_k(c,l-1,m,n)$, where $k \le c \le b$, in $O(b)$ operations. To compute R by repeated application of (19), we need l levels with $O(b)$ R's per level. This leads to time $O(lb^2)$ to compute a set of R. The time to compute F_l is dominated by the time needed to compute the R's, leading to time $O(lb^2)$ to compute a particular F. For fixed p and b and for all k, F_l can also be computed in time $O(lb^2)$.

5. Approximations.

5.1. Chernoff bounds. The sums for S_l and F_l are incomplete binomial sums. They do not have closed forms (implied by [15]), but, as we show below, Chernoff techniques [8] lead to useful approximations. For

$$L(i) = \begin{cases} 1, & i \le k, \\ 0, & i > k \end{cases} \quad \text{and} \quad U(i) = \begin{cases} 0, & i < k, \\ 1, & i \ge k \end{cases} \tag{20}$$

and for some fixed k in the range $0 \le k \le n$, we have

$$\sum_{0 \le i \le k} a_i = \sum_{0 \le i \le n} a_i L(i) \quad \text{and} \quad \sum_{k \le i \le n} a_i = \sum_{0 \le i \le n} a_i U(i). \tag{21}$$

In addition, when each $a_i \ge 0$, replacing $L(i)$ (or $U(i)$) with a pointwise upper bound gives an upper bound on the sum. Chernoff [8] noticed that useful bounds for partial binomial sums result when one uses

$$L(i) = x^{-k+i} \quad \text{with } x \le 1 \quad \text{and} \quad U(i) = x^{-k+i} \quad \text{with } x \ge 1 \tag{22}$$

and then chooses the x that gives the smallest upper bound.

The Chernoff bound for S is

$$S_l \le x^{-k} \sum_j \binom{b}{j} [xP(l)]^j [1 - P(l)]^{b-j} = x^{-k}[1 + (x-1)P(l)]^b \tag{23}$$

for any $x \ge 1$.

A Chernoff bound for R is

$$R_k(b,l,m,n) \le x^{-kl} \sum_{j_1,\ldots,j_l} \binom{b}{j_1, \ldots, j_l, b - j_1 - \cdots - j_l}$$

$$\times [xQ(m)]^{j_1+\cdots+j_l} [1 - P(m) - nQ(m)]^{b-j_1-\cdots-j_l} \tag{24}$$

$$\le x^{-kl}[1 - P(m) - (n - lx)Q(m)]^b \tag{25}$$

for any $x \ge 1$.

Using this Chernoff bound for R leads to the following Chernoff bound for F:

$$F_l \le y^{-k+1} \sum_j \binom{b}{j} [yP(l)]^j x^{-(k-j)l} [1 - P(l) + l(x-1)Q(l)]^{b-j} \tag{26}$$

$$\le x^{-kl} y^{-k+1} [1 + (x^l y - 1)P(l) + l(x-1)Q(l)]^b \tag{27}$$

for any $x \ge 1$ and any $y \le 1$.

5.2. Regions and boundaries for S_l. The optimum x for (23) is either on the boundary ($x = 1$) or when the derivative with respect to x is equal to zero. Letting x_* be the x value that gives a derivative of zero, we have

$$x_* = \frac{k[1 - P(l)]}{(b-k)P(l)}. \tag{28}$$

When $x_* \geq 1$, it is the optimum x for (23). Otherwise ($x_* < 1$) the optimum x is 1. In addition, we check whether x_* is strictly within the range ($x_* > 1$). This is the case when

$$k > bP(l). \tag{29}$$

This completes the first stage of finding the Chernoff approximation to S_l. In the next two sections we determine just how small the Chernoff bound is as a function of the parameters (b, k, l, and p). We show that the bound on S_l is an exponential function of the negative of the square of the distance α_1 (with $\alpha_1 = k/b - P(l)$) inside the boundary (29). As we vary k, S_l is extremely small inside the region ($k > bP(l)$), except near the boundary ($bP(l)$). Next we show that S_l is close to 1 once we go on the other side of the boundary; the difference between S_l and 1 is an exponential function of the negative of the square of the distance α_2 ($\alpha_2 = (P(l) - k/b) + 1/b = -\alpha_1 + 1/b$) from the boundary. Thus, knowing whether the optimizing x is strictly within range or not gives us the most basic information about S_l (whether it is small ($x_* > 1$) or large ($x_* < 1$)).

5.2.1. Upper bound on S_l. We now give an upper bound on S_l when $k > bP(l)$ to show that it is near 0. In the next section we give a lower bound when $k > bP(l)$ to show that in that case it is near 1.

By plugging the x_* value from (28) into the bound from (23) we obtain

$$S_l \leq \left(\frac{P(l)}{k}\right)^k \left(\frac{1 - P(l)}{b-k}\right)^{b-k} b^b \tag{30}$$

so long as $x_* \geq 1$. By (29) the condition $x_* > 1$ is equivalent to $k > bP(l)$, so we will define α_1 by

$$k = b[P(l) + \alpha_1]. \tag{31}$$

When k is greater than $bP(l)$, S_l goes to zero rapidly. In particular

$$S_l \leq e^{-b\alpha_1^2/\{2P(l)[1-P(l)]\} + O(b\alpha_1^3[1-P(l)]^{-2})} \tag{32}$$

when $\alpha_1 > 0$.

5.2.2. Lower bound on S_l. To obtain a lower bound on S_l when it is near 1, start with the right part of (11). Shift the relation between k and α_1 by 1 so that α_2 is defined implicitly by

$$k = b[P(l) - \alpha_2] + 1. \tag{33}$$

We can now modify the derivation of (32) (with $x_* < 1$) to obtain

$$S_l \geq 1 - e^{-b\alpha_2^2/\{2P(l)[1-P(l)]\} + O(b\alpha_2^3 P(l)^{-2})} \tag{34}$$

when $\alpha_2 > 0$ (the dominant O term here is different from what it was for (32)).

5.3. Regions and boundaries for F_l. To find the optimum value for x and y in (27) we start by taking derivatives of the bound with respect to x and y, setting each result to zero, and solving for x and y. We want the x_* that satisfies

$$(b-k)P(l)x_*^l y + (b-kl)Q(l)x_* - k[1-P(l)-lQ(l)] = 0. \tag{35}$$

We want the y_* that satisfies

$$(b-k+1)P(l)x^l y_* - (k-1)[1-P(l)+lQ(l)(x-1)] = 0. \tag{36}$$

When considering whether the optimum x and y are strictly within range ($x_* > 1$, $y_* < 1$) or on the boundary, there are four cases to investigate.

Region 1. Equation (35) with $y = 1$, $x_* > 1$.

Region 2. Equation (36) with $x = 1$, $y_* < 1$.

Region 3. Equations (35) and (36), $x_* > 1$, $y_* < 1$.

Region 4. $x = 1$, $y = 1$.

In (35) and (36), x_* is associated with the effectiveness of the candidacy test (inequality (25)), and y_* is associated with the probability of a set failing the frequency test (inequality (27)).

The main regions of interest are Region 1, where we will show that F_l is small because the candidacy test fails with high probability; Region 2, where F_l is small because S_l is near 1 (F_l can never be larger than $1 - S_l$ since failure requires not only passing the candidacy test but also failing the frequency test); and Region 4, where F_l has the trivial bound of 1. In section 5.3.3 we show that the set of parameter values that satisfy the conditions for Region 3 includes the intersection of Regions 1 and 2. Also Region 3 has no values outside of the union of Regions 1 and 2.

When the optimum value for at least one of x and y is strictly within range (not equal to 1) then the bound for F_l is smaller. It will be shown in section 5.4 that the bound on F_l is an exponential function of the square of the distance (basically the difference between k/b and $P(l)$ or $P(l-1)$; see section 5.4 for details) of x_* or y_* from the boundary, so F_l rapidly becomes extremely small as x_* or y_* moves away from the boundary.

5.3.1. Region 1. When $x_* > 1$, we will show in section 5.4.1 that the candidacy test fails with high probability. To find when this occurs, notice that (35) is satisfied by $x = 1$, $y = 1$ when

$$k = b[P(l) + Q(l)] = bP(l-1). \tag{37}$$

As b decreases, x_* increases. This implies that, for $y = 1$, $x_* > 1$ when

$$k > bP(l-1). \tag{38}$$

5.3.2. Region 2. When $y_* < 1$, we will show in section 5.4.2 that the frequency test succeeds with high probability. Consequently, since F_l can be no larger than $1 - S_l$, F_l is near 0 in this case.

When $x = 1$, the solution to (36) is

$$y_* = \frac{(k-1)[1-P(l)]}{(b-k+1)P(l)}. \tag{39}$$

This results in $y_* < 1$ when

$$k < bP(l) + 1. \tag{40}$$

For most parameter values, the regions (permitted values of k) of (38) and (40) do not overlap. However, subtracting the right side of (40) from the right side of (38), we find that they do overlap when

$$bQ(l) < 1. \tag{41}$$

This happens both when p^{l-1} is small ($p^{l-1} \leq 1/b$ is small enough) and also when $1-p$ is small ($1-p \leq 1/b$ is small enough). When (41) is true for all l, the Apriori Algorithm has no bad level. In this case, S_l is small for every l. Conditions where the Apriori Algorithm does have a bad level (cases where S_l is near 1) are discussed in section 5.4.4.

5.3.3. Region 3. To find values for the parameters such that $x_* > 1$ and $y_* < 1$ we need to satisfy (35) and (36) simultaneously. This results in the values

$$x_* = \frac{1 - P(l) - lQ(l)}{(b-k-l+1)Q(l)}, \tag{42}$$

$$y_* = (k-1)\left(\frac{(b-k-l+1)Q(l)}{1-P(l)-lQ(l)}\right)^{l-1}\frac{Q(l)}{P(l)}. \tag{43}$$

We have $x_* > 1$ when

$$k+l-1 < b < k+l-1+\frac{1-P(l)-lQ(l)}{Q(l)}. \tag{44}$$

The upper and lower limits are the same when $l = 1$, so the range is empty in that case.

All solutions to (44) are in the union of Regions 1 (inequality (38)) and 2 (inequality (40)). The smallest k that satisfies (38) is k just above $bP(l-1)$. This value for k satisfies (44) when

$$b < \frac{1}{Q(l)}. \tag{45}$$

Inequality (45) is true under the same conditions that (41) is true. Thus, (44) is satisfied by k values outside of Region 1 only when Regions 1 and 2 overlap. Since Region 1 gives a lower limit on k and Region 2 gives an upper limit, when Regions 1 and 2 overlap, their union includes all k values.

For $k = 1$, (43) implies that $y = 0$, which is less than 1. For $l = 1$, (43) has no solutions. For $k \geq 2$ and $l \geq 2$, (43) implies $y_* < 1$ when

$$b < k+l-1+\frac{1-P(l)-lQ(l)}{Q(l)}\left(\frac{P(l)}{(k-1)Q(l)}\right)^{1/(l-1)}. \tag{46}$$

For parameter values to be in Region 3, both (44) and (46) must be satisfied.

The upper bound on b from (46) is greater than the lower bound from (44). The upper bound on b from (46) is less than the upper bound from (44) when

$$k > \frac{1}{1-p}. \tag{47}$$

For $p < 1/2$, this condition is the same as $k > 1$.

Since

$$\frac{1-P(l)-lQ(l)}{Q(l)}\left(\frac{P(l)}{(k-1)Q(l)}\right)^{1/(l-1)} > 0, \tag{48}$$

any $k \geq b-l+1$ always satisfies (46). For $l=2$, this rightmost term from (46) reduces to

$$\frac{1}{k-1}, \tag{49}$$

which is less than 1 for $k \geq 2$. Thus, for $l=2$, the only solution to (46) is $k \geq b-l+1$.

The leftmost term of the right side of (46) (k) increases linearly with k; the rightmost term decreases with k. The rate of decrease slows down as k increases. As a result, the bound on b decreases at first and then increases. In some cases the bound (for fixed l) holds for small k, does not hold for moderate k, and then holds again for large k. As shown above (below (48)) the bound on b (inequality (44)) is always obeyed when k is large. Numerical investigations show that sometimes the bound also holds for small k, and sometimes it does not; sometimes the small k region extends all the way to the large k region, and sometimes it does not.

5.3.4. Region 4. In this case $x_* < 1$ and $y_* > 1$. Thus by section 5.3.1, $k < bP(l-1)$ and by section 5.3.2, $k > bP(l)+1$. Thus, for such k, we have $bP(l) < k < bP(l-1)$. In such cases, the candidacy test succeeds with probability near 1, but the frequency test fails with high probability (F_l is high). Thus, the Apriori Algorithm experiences a bad level in this region. In section 5.4.4 we give bounds on F_l.

5.4. Bounds for F_l. Section 5.3 found the parameters regions relevant to F_l. In this section we will establish bounds on F_l associated with these regions.

5.4.1. Bounds on F_l in Region 1. When $k > bP(l-1)$ we are in Region 1 of section 5.3. We now give an upper bound on F_l to show that it is near 0 in this case. Thus in this region the candidacy test fails with high probability.

By (27) with $y=1$

$$F_l \leq x_*^{-kl}[1+(x_*^l-1)P(l)+l(x_*-1)Q(l)]^b. \tag{50}$$

(Note that bounds on F_l obtained with $y=1$ are also bounds on $C_l = F_l + S_l$. The definition for C_l (equation (14)) has a sum over all values of j_0, but setting $y=1$ also sums at unit weight over all values of j_0.) The optimum x_* is given by (35). Solve (35) (with $y=1$) for x_* with $x_* = 1+\delta$ and small δ. Let θ stand for a function that approaches 1 in the limit as δ approaches 0. (Just as various big O's are associated with different implied constants, different θ's are associated with different functions that approach 1 in the limit.)

$$\delta = \frac{k-bP(l)-bQ(l)}{b[lP(l-1)]-kl[P(l-1)]}\left(1+\frac{[k-bP(l)-bQ(l)](b-k)l(l-1)P(l)\theta/2}{\{b[lP(l)+Q(l)]-kl[P(l-1)]\}^2}\right)^{-1}. \tag{51}$$

Define α_3 by

$$k = b[P(l-1)+\alpha_3]. \tag{52}$$

In (50) replace k by its value in terms of α_3 and plug in the value of x implied by (51) to obtain

$$F_l \le e^{-bl\theta\alpha_3^2/(2\{P(l-1)+(l-1)P(l)-l[P(l-1)]^2\})} \tag{53}$$

when α_3 is small enough, i.e.,

$$\alpha_3 = \{lP(l) + Q(l) - l[P(l-1)]^2\}o(1). \tag{54}$$

5.4.2. Region 2. When $k < bP(l) + 1$ we are in Region 2 of section 5.3, and by (34) nearly all item sets pass the frequency test. Since an item set must first pass the candidacy test and then fail the frequency test, F_l can be no larger than $1 - S_l$, which (by (34)) gives the bound

$$F_l \le e^{-b\alpha_2^2/\{2P(l)[1-P(l)]\}+O(\alpha_2^3 bP(l)^{-2})}, \tag{55}$$

where α_2 is defined by $k = b[P(l) - \alpha_2] + 1$ (equation (33)).

5.4.3. Region 3. Since the k values for Region 3 are entirely inside the union of Regions 1 and 2, we can use results from the previous two sections to obtain upper bounds on F_l. With additional algebra, even better upper bounds could be obtained, but the previous bounds are good enough for most purposes.

5.4.4. Region 4. When $bP(l) < k < bP(l-1)$ we are in Region 4 of section 5.3. The candidacy test succeeds with high probability, but the frequency test succeeds with low probability. We now give a lower bound on F_l to show that there are cases where it is near 1.

In (17), the quantity R_k is defined by sums where each $j_i \ge k$ (for $1 \le i \le l$). Using inclusion-exclusion arguments, an alternate way to compute R_k is

$$R_k(b,l,m,n) = \sum_h (-1)^h \binom{l}{h} r_k(b,l,m,n,h), \tag{56}$$

where

$$\begin{aligned} r_k(b,l,m,n,h) = {} & \sum_{\substack{j_1<k \\ j_2<k \\ \cdots \\ j_h<k \\ j_{h+1},\ldots,j_l}} \binom{b}{j_1, j_2, \ldots, j_l, b - j_1 - \cdots - j_l} \\ & \times [Q(m)]^{j_1+\cdots+j_l}[1 - P(m) - nQ(m)]^{b-j_1-\cdots-j_l} \end{aligned} \tag{57}$$

$$\begin{aligned} = {} & \sum_{\substack{j_1<k \\ j_2<k \\ \cdots \\ j_h<k}} \binom{b}{j_1, j_2, \ldots, j_l, b - j_1 - \cdots - j_l} \\ & \times [Q(m)]^{j_1+\cdots+j_h}[1 - P(m) - (n-l+h)Q(m)]^{b-j_1-\cdots-j_h}. \end{aligned} \tag{58}$$

The $h = 0$ term of (56) is the sum over the full range for the j's. The $h = 1$ term subtracts (for each j) the part of the range that is not included in the definition of R. The $h = 2$ corrects for the overcorrection of the $h = 1$ term (regions where two j's were out of range were subtracted off twice). Each successive h corrects for the previous h. Therefore, if the sum over h is terminated at some value before l, the

result is a lower or upper limit on R depending on whether the first omitted term is negative or positive. We use the following case of this result:

$$R_k(b,l,m,n) \geq r_k(b,l,m,n,0) - lr_k(b,l,m,n,1) \tag{59}$$
$$\geq [1-P(m)-(n-l)Q(m)]^b - l \times \sum_{j<k} \binom{b}{j} [Q(m)]^j [1-P(m)-(n-l+1)Q(m)]^{b-j}. \tag{60}$$

By (16) we have

$$F_l \geq \sum_{j_0<k} \binom{b}{j_0} [P(l)]^{j_0} \times \left([1-P(l)]^{b-j_0} - l \sum_{j<k-j_0} \binom{b-j_0}{j} [Q(l)]^j [1-P(l)-Q(l)]^{b-j_0-j} \right). \tag{61}$$

A lower bound on the sum that comes from the first term in the large parentheses can be obtained by combining (32) with the right part of (11). Applying reasoning similar to that leading to (27) gives the following bound related to the second term:

$$\sum_{j_0<k} \binom{b}{j_0} [P(l)]^{j_0} \sum_{j<k-j_0} \binom{b-j_0}{j} [Q(l)]^j [1-P(l)-Q(l)]^{b-j_0-j} \leq x^{-k+1} y^{-k+1} [1+(xy-1)P(l)+(x-1)Q(l)]^b, \tag{62}$$

with the requirement $x \leq 1$, $y \leq 1$. By setting $y = 1$ the bound becomes $x^{-k+1}[1 + (x-1)P(l-1)]^b$ (using (10)). The same techniques that give the exponential term in (34) can be applied to bound this term. Combining the bounds on the two parts gives

$$F_l \geq 1 - e^{-b\alpha_1^2/\{2P(l)[1-P(l)]\}+O(b\alpha_1^3[1-P(l)]^{-2})} - le^{-b\alpha_4^2/\{2P(l-1)[1-P(l-1)]\}+O(b\alpha_4^3[P(l-1)]^{-2})}, \tag{63}$$

with α_1 and α_4 related to k by $k = b[P(l)+\alpha_1]$ and $k = b[P(l-1)-\alpha_4]-1$ when both α_1 and α_4 are positive.

This bound is good enough to show that for some values of k, the Apriori Algorithm has one bad level. Consider k equal to the integer nearest $[bP(l)+bP(l-1)-1]/2$, i.e.,

$$k = \frac{bP(l)+bP(l-1)-1}{2} + \eta, \tag{64}$$

with $|\eta| \leq 1/2$. This results in

$$\alpha_1 = \frac{bP(l-1)-P(l)+1/b}{2} + \eta, \qquad \alpha_4 = \frac{P(l-1)-P(l)+1/b}{2} - \eta. \tag{65}$$

For fixed k, p, and l and for large b, α_1 and α_4 approach constants. The bound on F_l (inequality (61)) approaches 1 when b becomes large. Thus, when b is large, there is a k value where F_l is extremely close to 1. When k is near $bP(l)$ for some l the bound from (63) is not good enough to show that F_l is close to 1. The sample calculations (in section 7), however, show that for such k values there are usually two l values that are each moderately bad (F_l above a constant), at least when b is not small. Thus, the conclusion is that the Apriori Algorithm, when it is run on random data, usually has one bad level or two half-bad levels.

6. Total work. Equation (63) shows that for random data there are many cases where the Apriori Algorithm has one bad level, i.e., a level where many item sets pass the candidacy test but few of them pass the frequency test. Equation (41) shows that there are rare cases where the Apriori Algorithm has no bad levels. The Apriori Algorithm has a reputation for being effective in practice [2, 6, 20]. In this section we show that for many parameter values, even when there is a bad level, the bad level comes before the algorithm has done much work and the algorithm is extremely good for the levels after the bad one. This leads to good overall performance. Under the assumption that accesses to the original data dominate the running time, large running times result from those terms in (6) where the binomial coefficient is large and $S_l + F_l$ is not small. No algorithm that explicitly examines the data to verify the number of occurrences for a set can be fast if a large fraction of the possible sets for large l must be processed. The merit of the Apriori Algorithm is that $S_l + F_l$ usually becomes extremely small once l increases beyond the value that results in $k > bP(l)$. This is shown by the following rough calculation. Consider the ratio of the l and $l+1$ terms from (6):

$$\binom{|I|}{l+1} \Big/ \binom{|I|}{l} = \frac{|I|-l}{l+1} \approx \frac{|I|}{l+1}, \tag{66}$$

so long as l is much less than $|I|$. Now choose l so that k is near to $bP(l-1)$. Using this value of l in (53) results in α_3 near 0 and F_l near 1. Now consider the failure rate on the next level. Using l to refer to the earlier level, for the level following l, the bound on the failure rate is given by (53) using a value of α_3 near $b[P(l-1) - P(l)]$. For this value of α_3, (53) gives the bound

$$F_{l+1} \le e^{-b^3(l+1)[P(l-1)-P(l)]^2\theta/(2\{P(l)+lP(l+1)-(l+1)[P(l)]^2\})}. \tag{67}$$

Since $P(l)$ is p^l, for small p this bound is approximately

$$e^{-b^3(l+1)p^{-2}/2}. \tag{68}$$

Since almost every possible item set needs work on level l, the ratio of the amount of work that the Apriori Algorithm performs on level $l+1$ to the amount of work on level l is approximately

$$\frac{|I|}{l} e^{-(l+1)b^3/(2p^2)}. \tag{69}$$

In most interesting cases this ratio will be much less than 1. There is further improvement as l increases. For most parameter values and for random data the amount of work that the Apriori Algorithm does drops rapidly after the bad level.

7. Sample computations. This section contains sample calculations for $b = 1024$ baskets, $1 \le l \le 5$, with thresholds in the range $1 \le k \le 1024$. Table 1 gives S_l for $p = 1/2$, $1 \le l \le 5$. Table 2 gives S_l for $p = 1/16$. Table 3 gives F_l for $p = 1/2$. Table 4 gives F_l for $p = 1/16$. Each table has results for only a few selected values of k. The selected values for k include those were F_l is maximum, where it is just above $1/2$, and where it is just below $1/2$. Figure 1 is a graph of S_l for $p = 1/2$ (solid curves). Upper and lower bounds (dashed curves) from (32) and (34) are also included. Figure 2 is a graph of F_l (solid curve) for $p = 1/2$ along with the bounds from (53), (55), and (63) (dashed). For all bounds plotted in the figures, big O terms

TABLE 1
S_l for $b = 1024$, $p = 1/2$, and selected values of k.

k	S_1	S_2	S_3	S_4	S_5
1	$1-5.6\times10^{-309}$	$1-1.2\times10^{-128}$	$1-4.1\times10^{-60}$	$1-2.0\times10^{-29}$	$1-7.6\times10^{-15}$
2	$1-5.7\times10^{-306}$	$1-4.0\times10^{-126}$	$1-6.1\times10^{-58}$	$1-1.4\times10^{-27}$	$1-2.6\times10^{-13}$
3	$1-2.9\times10^{-303}$	$1-6.8\times10^{-124}$	$1-4.5\times10^{-56}$	$1-4.8\times10^{-26}$	$1-4.4\times10^{-12}$
4	$1-1\times10^{-300}$	$1-7.7\times10^{-122}$	$1-2.2\times10^{-54}$	$1-1.1\times10^{-24}$	$1-5.0\times10^{-11}$
5	$1-2.5\times10^{-298}$	$1-6.6\times10^{-120}$	$1-8.1\times10^{-53}$	$1-1.9\times10^{-23}$	$1-4.3\times10^{-10}$
22	$1-1.5\times10^{-265}$	$1-3.1\times10^{-95}$	$1-2.3\times10^{-34}$	$1-1.5\times10^{-10}$	$1-2.4\times10^{-2}$
32	$1-9.2\times10^{-250}$	$1-3.3\times10^{-84}$	$1-5.4\times10^{-27}$	$1-2.0\times10^{-6}$	5.2×10^{-1}
33	$1-2.9\times10^{-248}$	$1-3.4\times10^{-83}$	$1-2.4\times10^{-25}$	$1-4.3\times10^{-6}$	4.5×10^{-1}
45	$1-2.4\times10^{-231}$	$1-5.7\times10^{-72}$	$1-1.6\times10^{-19}$	$1-4.2\times10^{-3}$	1.6×10^{-2}
57	$1-6.8\times10^{-216}$	$1-3.1\times10^{-62}$	$1-3.7\times10^{-14}$	8.3×10^{-1}	3.1×10^{-5}
58	$1-1.2\times10^{-214}$	$1-1.7\times10^{-61}$	$1-9.2\times10^{-14}$	8.0×10^{-1}	1.6×10^{-5}
64	$1-1.9\times10^{-207}$	$1-4.0\times10^{-57}$	$1-1.4\times10^{-11}$	5.2×10^{-1}	2.5×10^{-7}
65	$1-2.9\times10^{-206}$	$1-2.0\times10^{-56}$	$1-3.0\times10^{-11}$	4.7×10^{-1}	1.2×10^{-7}
91	$1-6.2\times10^{-178}$	$1-1.9\times10^{-40}$	$1-1.1\times10^{-4}$	5.8×10^{-4}	2.3×10^{-18}
120	$1-1.5\times10^{-150}$	$1-7.6\times10^{-27}$	7.9×10^{-1}	5.3×10^{-11}	2.0×10^{-34}
121	$1-1.2\times10^{-149}$	$1-1.9\times10^{-26}$	7.6×10^{-1}	2.6×10^{-11}	4.7×10^{-35}
128	$1-1.3\times10^{-143}$	$1-9.8\times10^{-24}$	5.1×10^{-1}	1.4×10^{-13}	1.7×10^{-39}
129	$1-8.8\times10^{-143}$	$1-2.3\times10^{-23}$	4.8×10^{-1}	6.6×10^{-14}	3.8×10^{-40}
186	$1-2.8\times10^{-100}$	$1-7.1\times10^{-8}$	1.3×10^{-7}	8.9×10^{-39}	6.4×10^{-83}
247	$1-3.7\times10^{-65}$	7.5×10^{-1}	2.0×10^{-24}	1.2×10^{-75}	5.2×10^{-139}
248	$1-1.2\times10^{-64}$	7.3×10^{-1}	8.7×10^{-25}	2.4×10^{-76}	5.3×10^{-140}
256	$1-9.6\times10^{-61}$	5.1×10^{-1}	1.1×10^{-27}	7.2×10^{-82}	4.7×10^{-148}
257	$1-2.9\times10^{-60}$	4.8×10^{-1}	4.8×10^{-28}	1.4×10^{-82}	4.6×10^{-149}
377	$1-8.1\times10^{-18}$	3.8×10^{-17}	1.4×10^{-87}	9.8×10^{-182}	4.9×10^{-286}
503	7.2×10^{-1}	6.9×10^{-62}	1.5×10^{-178}	2.2×10^{-314}	2.1×10^{-458}
504	7.0×10^{-1}	2.4×10^{-62}	2.3×10^{-179}	1.5×10^{-315}	7.0×10^{-460}
512	5.1×10^{-1}	4.0×10^{-66}	4.4×10^{-186}	6.6×10^{-325}	9.3×10^{-472}
513	4.9×10^{-1}	1.3×10^{-66}	6.3×10^{-187}	4.4×10^{-326}	3.0×10^{-473}
533	1×10^{-1}	1.6×10^{-76}	3.3×10^{-204}	5.6×10^{-350}	1.9×10^{-503}

were ignored and θ was set to 1. The $p = 1/16$ cases do not lead to clear graphs, so none are given. For this case, one can best see what is happening by examining the tables.

When deciding which results to report, we had to balance the interest in large values for b (up to 100,000 in [2]) with the need to keep the computing time reasonable. Also, we had to balance the interest in small values for p with the need for results to show the various characteristics of the algorithm. And it is difficult to compute $(1-p)^j$ accurately when p is near zero and j is large. We used code where the number of multiplications increased only as fast as $\ln j$. The values of S were computed exactly using Maple and then converted to floating point. The Maple program was too slow to compute F in this way, so F was computed only with floating point arithmetic. For S, the results from exact arithmetic were not significantly different from those for floating point arithmetic, but the floating point calculations sometimes gave zero for values below 10^{-70}. Also, it was difficult to tell just how close to 1 a floating point value was once it went above $1 - 10^{-12}$.

From Table 1 and also from Figure 1, we see that, for fixed, moderate-sized values of k, S_l is extremely close to 1 for small values of l and that S_l is extremely small for large values of l. The transition from near 1 to small is quite sharp with increasing l. The transition value of l increases as k decreases. For large k, even S_1 is small. For small k one must go to large l values (not shown) before S_l becomes small. In Figure 1, the red curves (rightmost group) refer to $l = 1$. The upper dotted one is the upper

TABLE 2
S_l for $b = 1024$, $p = 1/16$, and selected values of k.

k	S_1	S_2	S_3	S_4	S_5
1	$1-2.0 \times 10^{-29}$	$1-1.8 \times 10^{-2}$	2.2×10^{-1}	1.6×10^{-2}	9.8×10^{-4}
2	$1-1.4 \times 10^{-27}$	$1-9.1 \times 10^{-2}$	2.6×10^{-2}	1.2×10^{-4}	4.8×10^{-7}
3	$1-4.8 \times 10^{-26}$	7.6×10^{-1}	2.2×10^{-3}	6.3×10^{-7}	1.5×10^{-10}
4	$1-1.1 \times 10^{-24}$	5.7×10^{-1}	1.3×10^{-4}	2.4×10^{-9}	3.8×10^{-14}
5	$1-1.9 \times 10^{-23}$	3.7×10^{-1}	6.6×10^{-6}	7.6×10^{-12}	7.3×10^{-18}
22	$1-1.5 \times 10^{-10}$	3.0×10^{-10}	3.2×10^{-35}	1.3×10^{-61}	4.2×10^{-88}
32	$1-2.0 \times 10^{-6}$	1.0×10^{-18}	1.0×10^{-55}	3.7×10^{-94}	1.1×10^{-132}
33	$1-4.3 \times 10^{-6}$	1.2×10^{-19}	7.3×10^{-58}	1.7×10^{-97}	3.1×10^{-137}
45	$1-4.2 \times 10^{-3}$	9.2×10^{-32}	2.0×10^{-84}	1.6×10^{-142}	1.1×10^{-192}
57	8.3×10^{-1}	2.5×10^{-45}	1.9×10^{-112}	5.5×10^{-181}	1.3×10^{-249}
58	8.0×10^{-1}	1.7×10^{-46}	7.8×10^{-115}	1.4×10^{-184}	2.1×10^{-254}
64	5.2×10^{-1}	8.9×10^{-54}	2.5×10^{-129}	2.6×10^{-206}	2.3×10^{-283}
65	4.7×10^{-1}	5.1×10^{-55}	8.9×10^{-132}	5.9×10^{-210}	3.3×10^{-288}
91	5.8×10^{-4}	2.0×10^{-89}	1.6×10^{-197}	5.1×10^{-307}	1.4×10^{-416}
120	5.3×10^{-11}	5.6×10^{-132}	4.8×10^{-275}	1.9×10^{-419}	6.1×10^{-564}
121	2.6×10^{-11}	1.6×10^{-133}	8.7×10^{-278}	2.1×10^{-423}	4.3×10^{-569}
128	1.4×10^{-13}	2.3×10^{-144}	4.5×10^{-297}	4.1×10^{-451}	3.1×10^{-605}
129	6.6×10^{-14}	6.3×10^{-146}	7.7×10^{-300}	4.4×10^{-455}	2.1×10^{-610}
186	8.9×10^{-39}	7.9×10^{-241}	1.8×10^{-463}	2.4×10^{-687}	2.6×10^{-911}
247	1.2×10^{-75}	1.0×10^{-352}	6.7×10^{-649}	3.0×10^{-945}	1.2×10^{-1243}
248	2.4×10^{-76}	1.3×10^{-354}	5.1×10^{-652}	1.5×10^{-950}	3.5×10^{-1249}
256	7.1×10^{-82}	5.4×10^{-370}	4.9×10^{-677}	3.3×10^{-985}	1.8×10^{-1293}
257	1.4×10^{-82}	6.3×10^{-372}	3.6×10^{-680}	1.5×10^{-989}	5.2×10^{-1299}

TABLE 3
F_l for $b = 1024$, $p = 1/2$, and selected values of k.

k	F_1	F_2	F_3	F_4	F_5
1	5.6×10^{-309}	1.1×10^{-128}	4.1×10^{-60}	2.0×10^{-29}	7.6×10^{-15}
2	5.7×10^{-306}	4.0×10^{-126}	6.1×10^{-58}	1.4×10^{-27}	2.6×10^{-13}
3	2.9×10^{-303}	6.8×10^{-124}	4.5×10^{-56}	4.8×10^{-24}	4.4×10^{-12}
4	1.0×10^{-300}	7.7×10^{-122}	2.2×10^{-54}	1.1×10^{-24}	5.0×10^{-11}
5	2.5×10^{-298}	6.6×10^{-120}	8.0×10^{-53}	1.9×10^{-23}	4.2×10^{-10}
22	1.5×10^{-265}	3.1×10^{-95}	2.3×10^{-34}	1.5×10^{-10}	2.4×10^{-2}
32	9.2×10^{-250}	3.3×10^{-83}	5.4×10^{-27}	2.0×10^{-6}	4.8×10^{-1}
33	2.9×10^{-248}	3.4×10^{-82}	2.4×10^{-26}	4.3×10^{-6}	5.5×10^{-1}
45	2.4×10^{-231}	5.7×10^{-72}	1.6×10^{-19}	4.2×10^{-3}	$1.0-3.4 \times 10^{-2}$
57	6.8×10^{-216}	3.1×10^{-62}	3.7×10^{-14}	1.7×10^{-1}	5.6×10^{-1}
58	1.2×10^{-214}	1.7×10^{-61}	9.2×10^{-14}	2.0×10^{-1}	5.0×10^{-1}
64	1.9×10^{-207}	4.0×10^{-57}	1.4×10^{-11}	4.8×10^{-1}	1.8×10^{-1}
65	2.9×10^{-206}	2.0×10^{-56}	3.0×10^{-11}	5.3×10^{-1}	1.4×10^{-1}
91	6.2×10^{-178}	1.9×10^{-40}	1.1×10^{-4}	$1.0-1.0 \times 10^{-3}$	5.6×10^{-7}
120	1.5×10^{-150}	7.6×10^{-27}	2.1×10^{-1}	5.2×10^{-1}	2.0×10^{-18}
121	1.2×10^{-149}	1.9×10^{-26}	2.4×10^{-1}	4.7×10^{-1}	6.8×10^{-19}
128	1.3×10^{-143}	9.8×10^{-24}	4.9×10^{-1}	1.9×10^{-1}	2.0×10^{-22}
129	8.8×10^{-143}	2.3×10^{-23}	5.2×10^{-1}	1.6×10^{-1}	6.1×10^{-23}
186	2.8×10^{-100}	7.1×10^{-8}	$1.0-3.4 \times 10^{-7}$	3.8×10^{-13}	7.3×10^{-60}
247	3.7×10^{-65}	2.5×10^{-1}	5.0×10^{-1}	4.7×10^{-40}	4.7×10^{-112}
248	1.2×10^{-64}	2.7×10^{-1}	4.7×10^{-1}	1.3×10^{-40}	4.7×10^{-113}
256	1.0×10^{-60}	4.9×10^{-1}	2.2×10^{-1}	4.5×10^{-45}	1.1×10^{-122}
257	2.9×10^{-60}	5.1×10^{-1}	1.9×10^{-1}	1.2×10^{-45}	3.9×10^{-124}
377	8.0×10^{-18}	1.0	8.7×10^{-30}	3.8×10^{-165}	
503	2.8×10^{-1}	5.2×10^{-1}	2.7×10^{-109}		
504	3.0×10^{-1}	4.9×10^{-1}	1.5×10^{-110}		
512	4.9×10^{-1}	2.6×10^{-1}	2.2×10^{-121}		
513	5.1×10^{-1}	2.4×10^{-1}	7.4×10^{-123}		
533	9.0×10^{-1}	1.0×10^{-2}	1.8×10^{-158}		

TABLE 4
F_l for $b = 1024$, $p = 1/16$, and selected values of k.

k	F_1	F_2	F_3	F_4	F_5
1	2.0×10^{-29}	1.8×10^{-2}	7.3×10^{-1}	1.9×10^{-3}	6.4×10^{-10}
2	1.4×10^{-27}	9.1×10^{-2}	7.3×10^{-1}	2.9×10^{-5}	6.3×10^{-13}
3	4.8×10^{-26}	2.4×10^{-1}	4.5×10^{-1}	2.3×10^{-7}	3.0×10^{-16}
4	1.1×10^{-24}	4.3×10^{-1}	2.0×10^{-1}	1.2×10^{-9}	9.8×10^{-20}
5	1.9×10^{-23}	6.3×10^{-1}	6.3×10^{-2}	4.7×10^{-12}	2.4×10^{-23}
22	1.5×10^{-10}	$1.0-6.0 \times 10^{-10}$	7.1×10^{-23}	3.9×10^{-61}	5.6×10^{-93}
32	2.0×10^{-6}	$1.0-4.1 \times 10^{-6}$	1.6×10^{-40}	1.7×10^{-93}	2.0×10^{-137}
33	4.3×10^{-6}	$1.0-8.7 \times 10^{-6}$	2.1×10^{-42}	8.1×10^{-97}	6.0×10^{-142}
45	4.2×10^{-3}	$1.0-8.4 \times 10^{-3}$	2.1×10^{-68}	1.1×10^{-137}	2.7×10^{-197}
57	1.7×10^{-1}	6.9×10^{-1}	2.4×10^{-92}	5.1×10^{-180}	1.2×10^{-274}
58	2.0×10^{-1}	6.4×10^{-1}	1.4×10^{-94}	1.3×10^{-183}	1.7×10^{-287}
64	4.8×10^{-1}	2.7×10^{-1}	3.4×10^{-108}	2.8×10^{-205}	
65	5.3×10^{-1}	2.2×10^{-1}	1.7×10^{-110}	6.4×10^{-209}	
91	$1-5.8 \times 10^{-4}$	3.3×10^{-7}	3.9×10^{-173}		
120	$1.0-5.3 \times 10^{-11}$	2.8×10^{-21}	4.0×10^{-250}		
121	$1.0-2.6 \times 10^{-11}$	6.7×10^{-22}	3.5×10^{-253}		
128	$1.0-1.4 \times 10^{-13}$	2.1×10^{-26}	9.7×10^{-276}		
129	$1.0-6.3 \times 10^{-14}$	4.4×10^{-27}	4.1×10^{-279}		
186	1.0	7.0×10^{-77}			
247	1.0	1.3×10^{-150}			
248	1.0	5.8×10^{-152}			
256	1.0	5.1×10^{-163}			
257	1.0	2.0×10^{-164}			

bound from (32). The solid one is the actual value from (11). The lower dashed one is the lower bound from (34). Proceeding to the left, we have corresponding groups for $l = 2$, 3, 4, and 5. For $l = 3$, 4, and 5, one can notice that the plotted "upper bound" goes below the actual value. This is because the big O term was omitted, and it is significant in these cases. Nonetheless, even without the big O term the upper bound gives the general idea for how the actual function behaves (it becomes extremely small). Table 2 shows that the $p = 1/16$ is similar to the $p = 1/2$ case. Notice that S with $p = 1/2$ and $l = 4$ has approximately the same value that S does for $p = 1/16$ and $l = 1$, particularly when k is small.

Table 3 and also Figure 2 show the values of F_l (solid) for $p = 1/2$ from (16), (17), and (19). Figure 3 also shows the upper (dotted) and lower (dashed) bounds computed from (53), (55), and (63). The red group of curves (rightmost) is for $l = 1$. The next rightmost group (orange) is for $l = 2$; the leftmost group is for $l = 5$ (blue). For any fixed k, there is one or sometimes two values of l for which F_l is not small. For most large values of k, there is just one l value where F_l is large, and for that one l value the resulting F_l is extremely close to 1, but for some large k values, there are two l values for which F_l is moderately large. As k becomes smaller, the l value that results in F_l being near 1 decreases. Also, F_l no longer becomes quite so close to 1. Table 4 shows F_l for $p = 1/16$. Notice that, for small k, F with $p = 1/2$ and $l = 4$ has approximately the same value as F does for $p = 1/16$ and $l = 1$. Table 5 shows the extend of the various regions when $b = 1024$, $p = 1/2$, and $1 \le l \le 5$. Table 6 shows information for the $p = 1/16$ case.

8. Experimental results. In this section we report the results of running the Apriori Algorithm on data that are more like those used in practice. We used three data sets: (1) synthetic data produced by the generator from the IBM Quest Research

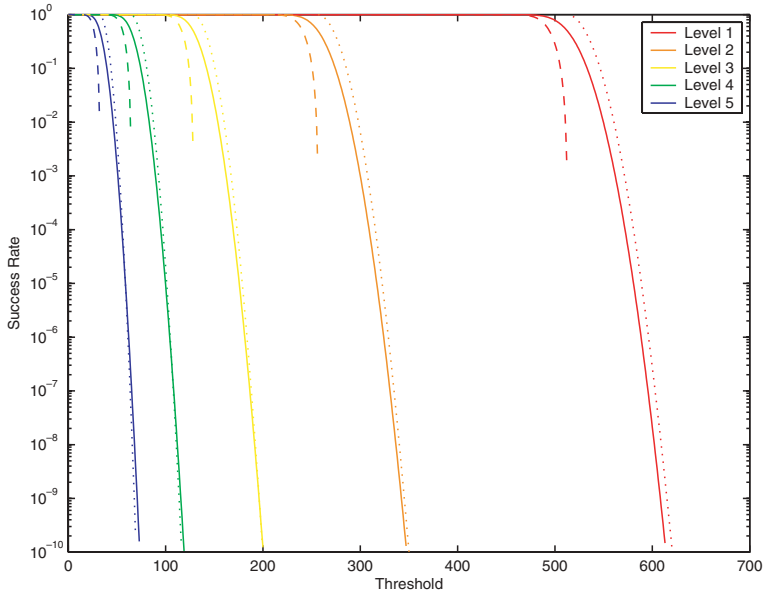


FIG. 1. *The value of S_l and approximations to S_l for $p = 1/2$ and $1 \leq l \leq 5$. The red curves are for S_1. The upper red dotted curve is the upper bound on S_1 ((32) with the big O term omitted). The solid red curve is the actual value of S_1. The lower red dashed curve is the lower bound on S_1 ((34) with the big O term omitted). The bounds are plotted only for the range where they are valid. Proceeding to the left, each group of three curves shows similar information on S_l for $l = 2$, 3, 4, and 5.*

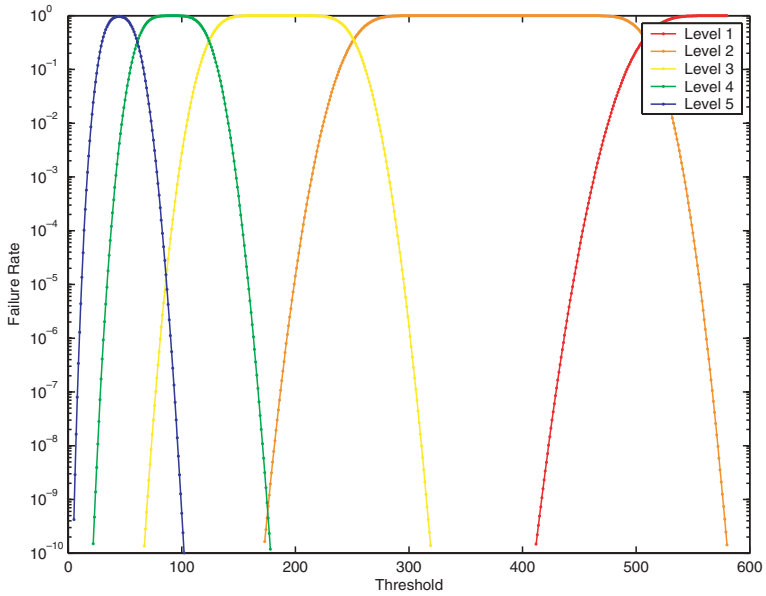


FIG. 2. *The value of F_l for $p = 1/2$ and $1 \leq l \leq 5$. The leftmost hump is the curve for $l = 5$, the next leftmost hump is for $l = 4$, etc.*

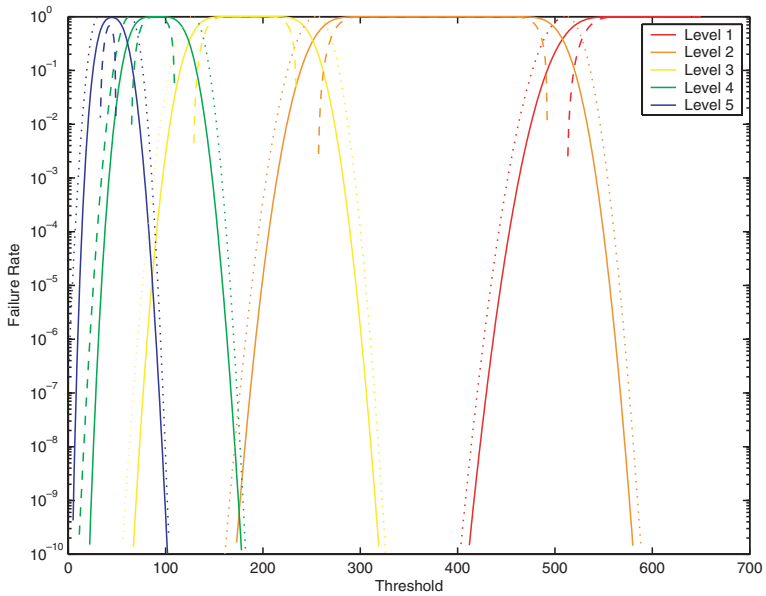


FIG. 3. *The value of F_l for $p = 1/2$ and $1 \leq l \leq 5$ along with upper (dotted) and lower (dashed) bounds.*

TABLE 5
Region boundaries for $b = 1024$, $p = 1/2$.

l	1	2	3	4	5
Region 1		$k \geq 513$	$k \geq 257$	$k \geq 129$	$k \geq 65$
Region 2	$k \leq 512$	$k \leq 256$	$k \leq 128$	$k \leq 64$	$k \leq 32$
Region 3		$1023 \leq k \leq 1023$	$1022 \leq k \leq 1022$	$1020 \leq k \leq 1021$	$1016 \leq k \leq 1020$
Region 4	$513 \leq k \leq 1024$	$257 \leq k \leq 512$	$129 \leq k \leq 256$	$65 \leq k \leq 128$	$33 \leq k \leq 64$

TABLE 6
Region boundaries for $b = 1024$, $p = 1/16$.

l	1	2	3	4	5
Region 1		$k \geq 65$	$k \geq 5$	$k \geq 1$	$k \geq 1$
Region 2	$k \leq 64$	$k \leq 4$	$k \leq 0$	$k \leq 0$	$k \leq 0$
Region 3		$1023 \leq k \leq 1023$	$1020 \leq k \leq 1022$	$833 \leq k \leq 1021$	$2 \leq k \leq 1020$
Region 4	$65 \leq k \leq 1024$	$5 \leq k \leq 64$	$\leq k \leq 4$		

Group [13]; (2) real data based on the U.S. Census [23]; and (3) real Web data [24]. These data sets have two major differences from the sets analyzed in the previous sections: (1) the probability of an item being in a basket is not the same for every item, but varies greatly from item to item (this effect is particularly strong in the census and Web data sets), and (2) the items are correlated [5].

Given the major differences between the data used in the analytical study and the data used for the experimental study, it is not surprising that many of the results from the experiments differ from those of the analysis. However, the experiments did verify one important conclusion of the analysis: work done by the Apriori Algorithm is dominated by the failure rate and the total failure rate is almost as high as it possibly can be. To be more precise, for most thresholds, there is a single level where most of

the work is done, and on this level almost every candidate fails to be a success.

In the random model, there is a fixed number of items, $|I|$, each occurring in a basket with probability p. For level l, when the threshold, k, is such that $k < bp^l$, almost every item set with l items ($\binom{|I|}{l}$ item sets) is frequent. Once l is large enough that $k > bp^l$, then almost no item set with l items is frequent. When p is small, the value of bp^l changes rapidly with l.

One can obtain an informal understanding of the experimental data presented below by permitting $|I|$, b, and p to vary (both with the threshold, k, and the level, l). In the random model, level l will have approximately $\binom{|I|}{l}$ successful item sets so long as k is much less than bp^l, and it will have a lot less if k is much larger than bp^l. Also, when p is small, the value of bp^l changes rapidly with l. The ratio of the number of item sets on level l to level $l-1$ is $\binom{|I|}{l}/\binom{|I|}{l-1}$, which is approximately $|I|/l$ if $|I|$ is much larger than l. Thus, we would expect the number of item sets to increase rapidly with l until the level where bp^l becomes larger than k. Even when the effective $|I|$ decreases with l and the effective p increases, one is likely to get behavior that is qualitatively the same. The number of item sets will increase rapidly at first, and then it will suddenly decrease. The cause of the decrease will be the high failure rate on the level just before the decrease. This will be the level where most of the work is done, because it is the level with the most candidates.

Now, consider some actual data sets. These data sets were run through a version of the Apriori Algorithm that computes frequent item sets for all values of the threshold with a lower limit being imposed on the threshold as needed to avoid running out of computer memory. Figures 4–9 show the results for three data sets. In each figure, the bottom axis shows the threshold. Each color curve refers to one level of the Apriori Algorithm: level 1, red; level 2, orange; level 3, yellow; level 4, green; level 5, blue. Figures 4, 6, and 8 show, for each of the data sets, the number of candidates as a solid curve, the upper bound from [10] as a dotted curve, and the number of failures as a dashed curve. For a given level, the upper bound is highest, the number of candidates is in the middle, and the number of failures is lowest. Often the dotted and/or dashed curves are so close to the solid curve that they don't show up. Figures 5, 7, and 9 show, for each of the data sets, the ratio of failures to candidates (for each level).

8.1. Synthetic data. The first data set was generated using the synthetic data set generator from the IBM Quest Research Group [13] to create data sets in the same fashion as [3]. We tested several synthetic data sets, each of which had 1000 items. We report the results for the T5.I2.D100K (average transaction size, 5; size of the average maximal frequent item set, 2; number of transactions, 100,000), as they are representative of the experimental results for the other synthetic data sets.

The solid curves in Figure 4 show the number of candidates on each level (which is proportional to the work done on the level). Notice that for high thresholds (above 849) the most work is done on level 1; for intermediate thresholds (between 16 and 849) the most work is done on level 2; and for low thresholds (between 1 and 15) the most work is done on level 3. Notice that for the highest solid curve (which one is highest depends on the threshold) the number of failures is almost equal to the number of candidates. In Figure 4 this shows up by the dashed curve falling on top of the solid curve (or almost on top). Figure 5 shows the ratio of failures to candidates. This ratio is close to 1 for the curve that is uppermost in Figure 4.

For each threshold, the uppermost curve represents the bad level, the level where most of the work is done and where almost every candidate is a failure. For levels past the bad level, one may want to use a version of the Apriori Algorithm that counts

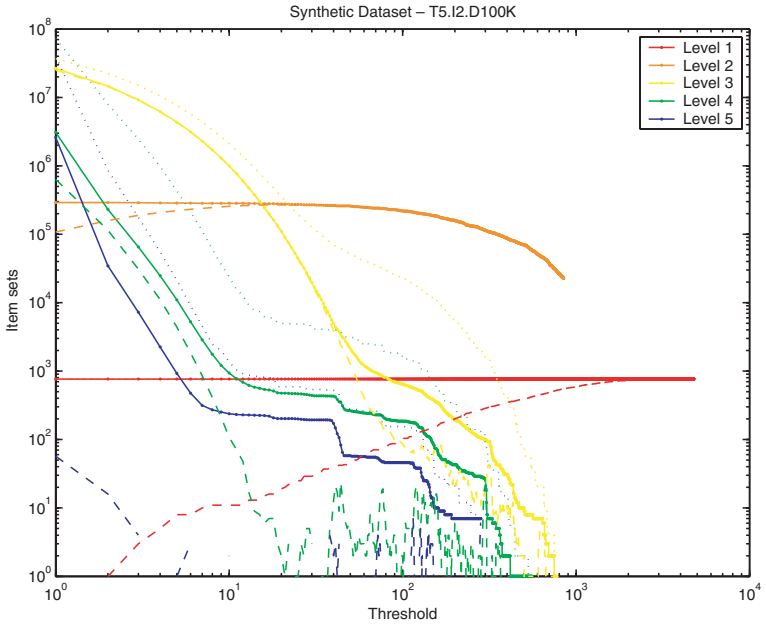


FIG. 4. *Behavior of the algorithm on synthetic data. The dotted lines show the upper bounds, the solid lines show the actual number of candidates considered, and the dashed lines show the number of candidates that fail for each level.*

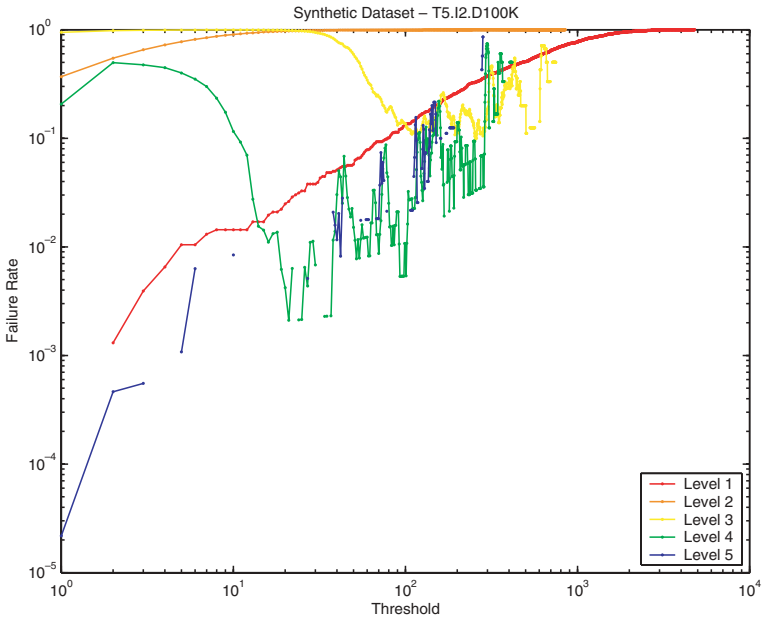


FIG. 5. *The failure rate for the synthetic data set.*

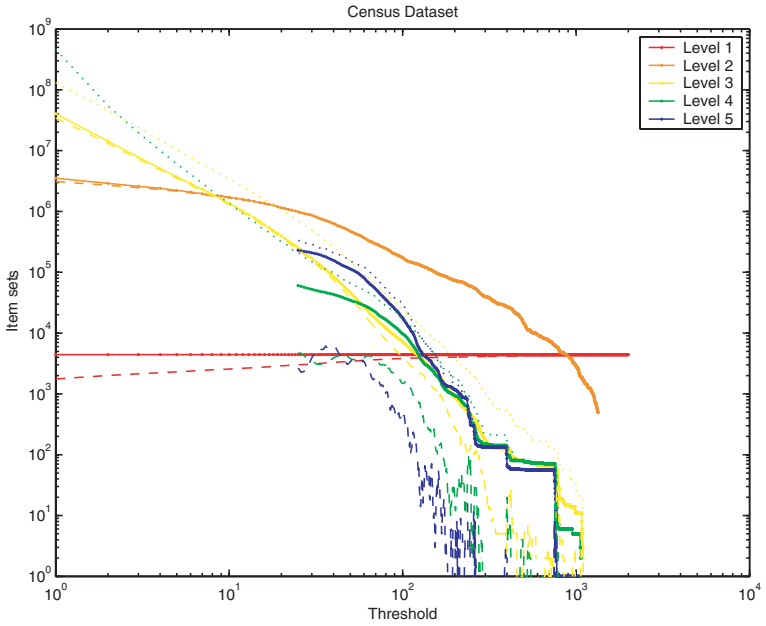


FIG. 6. *Behavior of the algorithm on census data. The dotted lines show the upper bounds, the solid lines show the actual number of candidates considered, and the dashed lines show the number of candidates that fail for each level.*

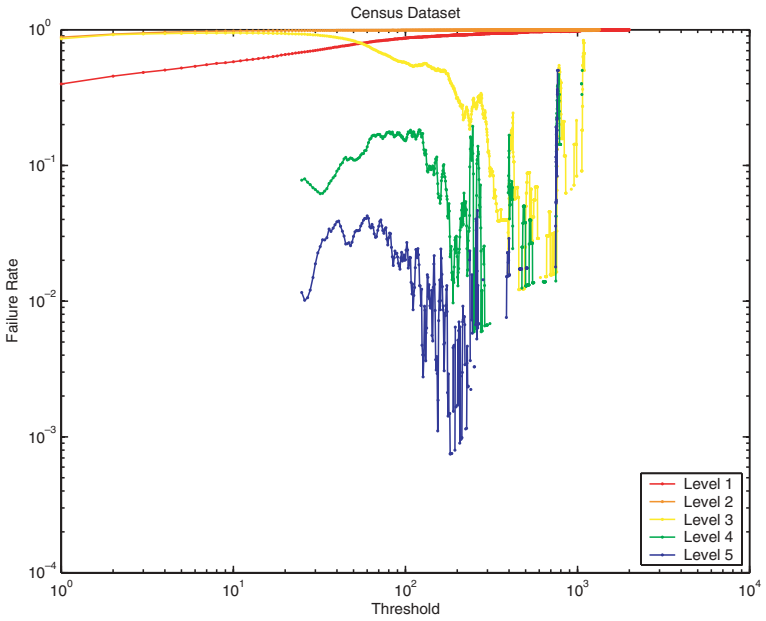


FIG. 7. *The failure rate for the census data set.*

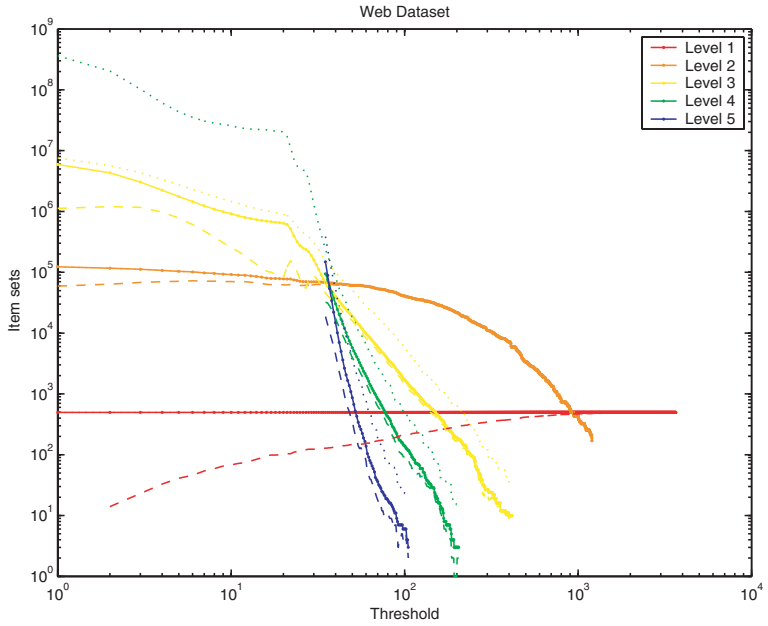


FIG. 8. *Behavior of the algorithm on Web data. The dotted lines show the upper bounds, the solid lines show the actual number of candidates considered, and the dashed lines show the number of candidates that fail for each level.*

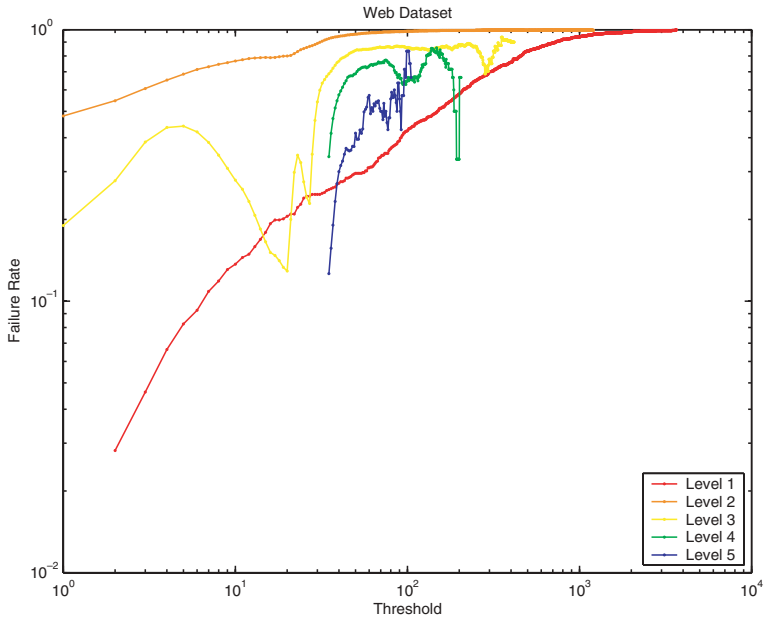


FIG. 9. *The failure rate for the Web data set. Note that each of the first three levels is "bad" for certain ranges of the threshold value.*

item sets for all remaining levels in one pass. (This was one of the main motivations for [10].)

8.2. Census data. The second data set was U.S. Census data, using Public Use Microdata Samples (PUMS) [23]. We chose the same sample that was used by [5, 6]. The data represents a 5% sample of the 1990 U.S. Census data for Washington, D.C. It contains 30,370 entries with 122 attributes.

Following [5, 6, 11], we modified the data in the following ways. For monetary values, we took the ceiling of the logarithm of the value. Then we assigned a unique integer to each possible value in the data. In total, the converted PUMS data was 16.6 MB, with 7523 unique items. In addition, we pruned the highly frequent items, which yield an intractable number of frequent item sets.

Due to the high number of correlated items in this type of data our computing resources did not permit us to compute levels 4 and 5 for thresholds less than 25.

The solid curves in Figure 6 show the number of candidates on each level. Notice that for high thresholds (above 872) the most work is done on level 1 and for intermediate thresholds (between 25 and 872) the most work is done on level 2. We are missing data for low thresholds. Figures 6 and 7 show that for the level where most of the work is done, the ratio of failures to candidates is almost 1.

8.3. Web data. The third data set was the BMS-WebView-1 data set described in [24]. This data contains click stream data from a Web-based merchandising company. There are 59,602 transaction and 497 distinct items in the Web data.

Figure 8 shows the results for the Web data. Resources did not permit thresholds below 35 for levels 4 and above.

Level 3, however, has a large enough number of successes to suggest that levels 4 and 5 are the truly "bad" levels for this data. The slope of these two levels is extremely steep, with level 5 being nearly vertical. Figure 9 shows the ratio of failures to candidates for the Web data.

9. Discussion. For random shopper and for most values of the parameters, the amount of work that the Apriori Algorithm does increases rapidly with the level until a bad level is reached. Until the bad level, almost every item set is frequent; level l will have almost $\binom{|I|}{l}$ successes. At the bad level and beyond almost every item set fails to be frequent. Almost all of the work done by the algorithm consists of processing the failures on the bad level. Which level is bad depends on the threshold (and the other parameters). For special values of the threshold two neighboring level will each be halfway bad.

The experimental data in the paper came from shoppers that were far from random. For most thresholds, there still was a bad level where almost all the work for the whole algorithm consisted of processing failures on the bad level. The fraction of failures on the early levels was not large, but it was large enough that the number of successes on level l was much less than $\binom{|I|}{l}$.

Our assumptions on random shoppers and the assumptions that realize the worst-case bounds of [10, Theorem 1] (highly correlated shopping) appear quite different, but in some ways they are quite similar. Consider the case where the number of successes on level l is $\binom{m}{l}$ for some integer m. Then the upper bound limit on the number of candidates for level $l+1$ is $\binom{m}{l+1}$, the same as for the random shopper case when $p = 1$ and $|I| = m$. Indeed the random shopper model gives essentially the same prediction for the number of candidates for any value of p large enough that $k < bp^l$. From this point of view, the key insight of their Theorem 1 is that the

binomial coefficient leads to a useful estimate of how many items are still important when the Apriori Algorithm gets to level l.

We believe that comparing properties that are observed for the random shopper with the properties that are observed with worst-case shoppers can give some insight as to how the Apriori Algorithm will behave with typical data sets.

Appendix. This section gives proofs and the derivations of equations.

Proof. A subpart of the Apriori Algorithm is NP-complete. Determining whether some set of size l occurs k times is in NP because one can guess the set and then verify the number of occurrence by counting the occurrences. The proof that the problem is NP-hard uses reduction from the balanced complete bipartite subgraph problem: given a positive integer K and a bipartite graph with vertices V and edges E, determine whether there are two disjoint sets of edges (V_1 and V_2) such that $|V_1| = K$, $|V_2| = K$, and such that there is an edge in E between each vertex in V_1 and each vertex in V_2. Since any such subgraph must be in a single connected component of the original graph, we can process each connected component of the graph separately. In a single connected component, the vertices of a bipartite graph naturally fall into two groups where all the edges in a group are connected by paths of even length. To map the given single component bipartite graph to baskets and items, associate (in a one-to-one manner) each vertex of one part with an item, and associate (in a one-to-one manner) each vertex of the other part with a basket. Have item i in basket b if and only if the vertex associated with i has an edge connecting to the vertex associated with b. If there is solution to the given instance of the balanced complete bipartite subgraph problem, then that solution directly gives an item set of size K that occurs in K baskets. Also if there is an item set of size K that occurs in K baskets, then the corresponding subgraph is a solution to the given instance. □

Equation (11). We have j ($j \geq k$) baskets that contain the set and $b - j$ baskets that do not contain the set, so

$$S_l = \sum_{j \geq k} \binom{b}{j} [P(l)]^j [1 - P(l)]^{b-j}. \tag{11a}$$

From the binomial theorem we have

$$\sum_{j} \binom{b}{j} [P(l)]^j [1 - P(l)]^{b-j} = 1, \tag{A1}$$

so

$$\sum_{j \geq k} \binom{b}{j} [P(l)]^j [1 - P(l)]^{b-j} = 1 - \sum_{j < k} \binom{b}{j} [P(l)]^j [1 - P(l)]^{b-j}. \tag{11b}$$

Equation (12). The factor $[P(l)]^{j_0}$ is the probability that the first j_0 baskets obey condition M_0, $[Q(l)]^{j_1}$ is the probability that the next j_1 baskets obey condition M_1, ..., $[Q(l)]^{j_l}$ is the probability that the next j_l baskets obey condition M_l, and $[1 - P(l) - lQ(l)]^{b-j_0-\cdots-j_l}$ is the probability that the remaining baskets obey none of the conditions $M_0, \ldots, M_l$. (Notice that each basket obeys at most one of the conditions M_h ($0 \leq h \leq l$).) However, the various baskets can come in any order, and the multinomial coefficient allows for this. Thus the probability that j_h baskets obey condition M_h ($0 \leq h \leq l$) and that the remaining $b - j_0 - \cdots - j_l$ baskets do not obey

any of the conditions is

$$\binom{b}{j_0,\ldots,j_l,b-j_0-\cdots-j_l}[P(l)]^{j_0}[Q(l)]^{j_1+\cdots+j_l}[1-P(l)-lQ(l)]^{b-j_0-\cdots-j_l}. \tag{12}$$

Equation (18).

$$\begin{aligned} R_k(b,l,m,n) &= \sum_{\substack{j_1\ge k\\ j_2\ge k\\ \cdots\\ j_l\ge k}} \binom{b}{j_1,\ldots,j_l,b-j_1-\cdots-j_l} \\ &\quad\times [Q(m)]^{j_1+\cdots+j_l}[1-P(m)-nQ(m)]^{b-j_1-\cdots-j_l} \qquad (17)\\ &= \sum_{j_1\ge k}\binom{b}{j_1}[Q(m)]^{j_1}\sum_{\substack{j_2\ge k\\ j_3\ge k\\ \cdots\\ j_l\ge k}}\binom{b-j_1}{j_2,\ldots,j_l,b-j_1-\cdots-j_l} \\ &\quad\times [Q(m)]^{j_2+\cdots+j_l}[1-P(m)-nQ(m)]^{(b-j_1)-j_2-\cdots-j_l} \qquad (A2)\\ &= \sum_{j\ge k}\binom{b}{j}[Q(m)]^{j}R_k(b-j,l-1,m,n). \qquad (18) \end{aligned}$$

Equation (19). The boundary condition is just (17) with l replaced with 0.

Equation (27). Using the binomial theorem on the j_l sum in (25), we have

$$\begin{aligned} &x^{-kl}\sum_{j_1,\ldots,j_l}\binom{b}{j_1,\ldots,j_l,b-j_1-\cdots-j_l} \\ &\qquad\times [xQ(m)]^{j_1+\cdots+j_l}[1-P(m)-nQ(m)]^{b-j_1-\cdots-j_l} \\ &= x^{-kl}\sum_{j_1,\ldots,j_{l-1}}\binom{b}{j_1,\ldots,j_{l-1},b-j_1-\cdots-j_{l-1}} \\ &\qquad\times [xQ(m)]^{j_1+\cdots+j_{l-1}}[1-P(m)-nQ(m)]^{b-j_1-\cdots-j_{l-1}}. \qquad (A3) \end{aligned}$$

The remaining $l-1$ sums can be done the same way to obtain

$$\begin{aligned} R_k(b,l,m,n) &\le x^{-kl}\sum_{j_1,\ldots,j_l}\binom{b}{j_1,\ldots,j_l,b-j_1-\cdots-j_l} \\ &\qquad\times [xQ(m)]^{j_1+\cdots+j_l}[1-P(m)-nQ(m)]^{b-j_1-\cdots-j_l} \\ &\le x^{-kl}[1-P(m)-(n-lx)Q(m)]^b. \qquad (25) \end{aligned}$$

Equation (28). Less algebra is needed to minimize the logarithm of the bound, and it leads to the same result. Start with the derivative of the logarithm of (23):

$$\frac{d\{-k\ln x+b\ln[1+(x-1)P(l)]\}}{dx} = \frac{-k}{x}+\frac{bP(l)}{1+P(l)(x-1)}. \tag{A4}$$

Now set the logarithm to zero and replace x (the free variable) with x_* (the value

that results in the derivative being zero):

$$\frac{-k}{x_*} + \frac{bP(l)}{1 + P(l)(x_* - 1)} = 0, \tag{A5}$$

$$-kP(l)(x_* - 1) - k + bP(l)x_* = 0, \tag{A6}$$

$$x_* = \frac{k[1 - P(l)]}{(b - k)P(l)}. \tag{28}$$

Equation (29).

$$\frac{k[1 - P(l)]}{(b - k)P(l)} > 1, \tag{A7}$$

$$k[1 - P(l)] > (b - k)P(l) \tag{A8}$$

(since $b > k$), so

$$k > bP(l). \tag{29}$$

Equation (30). Plugging (28) into (23) gives

$$S_l \le \left(\frac{k[1 - P(l)]}{(b - k)P(l)}\right)^{-k} \left[1 + \left(\frac{k[1 - P(l)]}{(b - k)P(l)} - 1\right) P(l)\right]^b, \tag{A9}$$

$$S_l \le \{k[1 - P(l)]\}^{-k}[P(l)]^k(b - k)^{-b+k}(b - k + \{k[1 - P(l)] - (b - k)P(l)\})^b,$$

$$S_l \le \left(\frac{P(l)}{k}\right)^k \left(\frac{1 - P(l)}{b - k}\right)^{b-k} b^b. \tag{30}$$

Equation (32). Replace k in (30) with its value in terms of α_1 (equation (31)):

$$S_l \le \left(\frac{P(l)}{b[P(l) + \alpha_1]}\right)^{b[P(l)+\alpha_1]} \left(\frac{1 - P(l)}{b - b[P(l) + \alpha_1]}\right)^{b-b[P(l)+\alpha_1]} b^b, \tag{A10}$$

$$S_l \le \left(\frac{P(l)}{b[P(l) + \alpha_1]}\right)^{b[P(l)+\alpha_1]} \left(\frac{1 - P(l)}{b[1 - P(l) - \alpha_1]}\right)^{b[1-P(l)-\alpha_1]} b^b, \tag{A11}$$

$$S_l \le \left(\frac{1}{b[1 + \alpha_1/P(l)]}\right)^{b[P(l)+\alpha_1]} \left(\frac{1}{b\{1 - \alpha_1/[1 - P(l)]\}}\right)^{b[1-P(l)-\alpha_1]} b^b, \tag{A12}$$

$$S_l \le \left(\frac{1}{1 + \alpha_1/P(l)}\right)^{b[P(l)+\alpha_1]} \left(\frac{1}{1 - \alpha_1/[1 - P(l)]}\right)^{b[1-P(l)-\alpha_1]}. \tag{A13}$$

To further simplify this, we will write it as $S_l \le e^X$ with

$$X = \ln\left[\left(\frac{1}{1 + \alpha_1/P(l)}\right)^{b[P(l)+\alpha_1]} \left(\frac{1}{1 - \alpha_1/[1 - P(l)]}\right)^{b[1-P(l)-\alpha_1]}\right] \tag{A14}$$

$$= -b[P(l) + \alpha_1] \ln\left(1 + \frac{\alpha_1}{P(l)}\right) - b[1 - P(l) - \alpha_1] \ln\left(1 - \frac{\alpha_1}{[1 - P(l)]}\right). \tag{A15}$$

Dividing by b, we have

$$\frac{X}{b} = -[P(l)+\alpha_1]\ln\left(1+\frac{\alpha_1}{P(l)}\right) - [1-P(l)-\alpha_1]\ln\left(1-\frac{\alpha_1}{[1-P(l)]}\right) \tag{A16}$$

$$= -[P(l)+\alpha_1]\left[\left(\frac{\alpha_1}{P(l)}\right) - \frac{1}{2}\left(\frac{\alpha_1}{P(l)}\right)^2 + O\left(\left(\frac{\alpha_1}{P(l)}\right)^3\right)\right] \tag{A17}$$

$$+ [1-P(l)-\alpha_1]\left[\left(\frac{\alpha_1}{1-P(l)}\right) + \frac{1}{2}\left(\frac{\alpha_1}{1-P(l)}\right)^2 + O\left(\left(\frac{\alpha_1}{1-P(l)}\right)^3\right)\right] \tag{A18}$$

$$= -\alpha_1 + \frac{\alpha_1^2}{2P(l)} - O\left(\frac{\alpha_1^3}{P(l)^2}\right) - \frac{\alpha_1^2}{P(l)} + \frac{\alpha_1^3}{2P(l)^2} - O\left(\frac{\alpha_1^4}{P(l)^3}\right)$$

$$+ \alpha_1 + \frac{\alpha_1^2}{2[1-P(l)]} + O\left(\frac{\alpha_1^3}{[1-P(l)]^2}\right) - \frac{\alpha_1^2}{1-P(l)} - \frac{\alpha_1^3}{2[1-P(l)]^2}$$

$$- O\left(\frac{\alpha_1^4}{[1-P(l)]^3}\right) \tag{A19}$$

$$= -\frac{\alpha_1^2}{2P(l)} - \frac{\alpha_1^2}{2[1-P(l)]} + O\left(\frac{\alpha_1^3}{[1-P(l)]^2}\right) - O\left(\frac{\alpha_1^3}{P(l)^2}\right). \tag{A20}$$

The big O is with respect to α_1. We assume that $0 < p < 1$. Since negative big O terms can be dropped in an upper limit,

$$S_l \le e^{-b\alpha_1^2/\{2P(l)[1-P(l)]\}+O(b\alpha_1^3[1-P(l)]^{-2})}. \tag{32}$$

Equation (35). The derivative of the logarithm of the bound on F (inequality (27)) with respect to x is

$$\frac{d\,[-kl\ln x - (k-1)\ln y + b\ln[1+(x^l y-1)P(l)+l(x-1)Q(l)]}{dx}$$

$$= \frac{-kl}{x} + \frac{b[lx^{l-1}yP(l)+lQ(l)]}{1+(x^l y-1)P(l)+l(x-1)Q(l)}. \tag{A21}$$

Setting this to zero gives

$$-kl[1+(x_*^l y-1)P(l)+l(x_*-1)Q(l)] + bx_*[lx_*^{l-1}yP(l)+lQ(l)] = 0, \tag{A22}$$

$$-kl - klP(l)x_*^l y + klP(l) + kl^2Q(l) - kl^2Q(l)x_* + blP(l)x_*^l y + blQ(l)x_* = 0, \tag{A23}$$

$$(b-k)P(l)x_*^l y + (b-kl)Q(l)x_* - k[1-P(l)-lQ(l)] = 0. \tag{35}$$

Equation (36). The derivative of the logarithm bound (inequality (27)) with respect to y gives

$$\frac{d\,[-kl\ln x - (k-1)\ln y + b\ln[1+(x^l y-1)P(l)+l(x-1)Q(l)]}{dy} \tag{A24}$$

$$= -\frac{k-1}{y} + \frac{bx^lP(l)}{1+(x^l y-1)P(l)+l(x-1)Q(l)}. \tag{A25}$$

Setting this to zero gives

$$-(k-1)[1+(x^l y_*-1)P(l)+l(x-1)Q(l)]+bx^l y_* P(l)=0, \tag{A26}$$

$$(b-k+1)P(l)x^l y_*-(k-1)[1-P(l)+lQ(l)(x-1)]=0. \tag{36}$$

Equation (37). Equation (35) with $x_*=1$, $y=1$ is

$$(b-k)P(l)+(b-kl)Q(l)-k[1-P(l)-lQ(l)]=0, \tag{A27}$$

$$k=b[P(l)+Q(l)]=bP(l-1). \tag{37}$$

Equation (38). By implicit differentiation of (35) (with $y=1$) we have

$$\frac{d\{(b-k)P(l)x_*^l+(b-kl)Q(l)x_*-k[1-P(l)-lQ(l)]\}}{db}=0, \tag{A28}$$

$$P(l)x_*^l+Q(l)x_*+[(b-k)lP(l)x_*^{l-1}+(b-kl)Q(l)]\frac{d\,x_*}{db}=0, \tag{A29}$$

$$\frac{d\,x_*}{db}=-\frac{P(l)x_*^l+Q(l)x_*}{(b-k)lP(l)x_*^{l-1}+(b-kl)Q(l)}. \tag{A30}$$

Since x_* solves (35) (with $y=1$) we have

$$(b-k)P(l)x_*^l+(b-kl)Q(l)x_*=k[1-P(l)-lQ(l)], \tag{A31}$$

which is positive. Thus

$$\begin{aligned}(b-k)P(l)lx_*^{l-1}+(b-kl)Q(l)=&\frac{(b-k)P(l)x_*^l+(b-kl)Q(l)x_*}{x_*}\\&+(l-1)(b-k)P(l)x_*^{l-1}\end{aligned} \tag{A32}$$

is also positive (because $x_*>0$, $b-k>0$, and $l\geq 1$). Thus, $d\,x_*/db$ is negative. If we start at the b value that results in $x_*=1$, and decrease b, then x_* increases. Thus, it becomes larger than 1 and stays larger than 1.

Equation (39). Equation (36) with $x=1$ is

$$(b-k+1)P(l)y_*-(k-1)[1-P(l)]=0, \tag{A33}$$

$$y_*=\frac{(k-1)[1-P(l)]}{(b-k+1)P(l)}. \tag{39}$$

Equation (40).

$$\frac{(k-1)[1-P(l)]}{(b-k+1)P(l)}<1, \tag{A34}$$

$$(k-1)[1-P(l)]<(b-k+1)P(l), \tag{A35}$$

$$k-1<bP(l), \tag{A36}$$

$$k<bP(l)+1. \tag{40}$$

Equation (42). From (35)

$$P(l)x_*^l y = \frac{k[1-P(l)-lQ(l)]-(b-kl)Q(l)x_*}{b-k}. \tag{A37}$$

From (36)

$$P(l)x^l y_* = \frac{(k-1)[1-P(l)+l(x-1)Q(l)]}{b-k+1}. \tag{A38}$$

Setting x to x_*, y to y_*, the two right sides equal, and clearing fractions gives

$$\begin{aligned} &k(b-k+1)[1-P(l)-lQ(l)]-(b-kl)(b-k+1)Q(l)x \\ &\quad = (k-1)(b-k)[1-P(l)+lQ(l)x-lQ(l)], \end{aligned} \tag{A39}$$

$$\begin{aligned} &(b^2-bk+b-bkl+k^2l-kl)Q(l)x = (bk-k^2+k)[1-P(l)-lQ(l)] \\ &\quad -(bk-k^2-b+k)[1-P(l)+lQ(l)x-lQ(l)], \end{aligned} \tag{A40}$$

$$\begin{aligned} &(b^2-bk+b-bkl+k^2l-kl+bkl-k^2l-bl+kl)Q(l)x \\ &\quad = (bk-k^2+k-bk+k^2+b-k)[1-P(l)-lQ(l)], \end{aligned} \tag{A41}$$

$$(b^2-bk+b-bl)Q(l)x = b[1-P(l)-lQ(l)], \tag{A42}$$

$$x = \frac{1-P(l)-lQ(l)}{(b-k-l+1)Q(l)}. \tag{42}$$

Equation (43). Plugging the value of x from (42) into (35) gives

$$\begin{aligned} &(b-k)\left(\frac{1-P(l)-lQ(l)}{(b-k-l+1)Q(l)}\right)^l P(l)y \\ &\quad + \frac{(b-kl)[1-P(l)-lQ(l)]}{b-k-l+1} - k[1-P(l)-lQ(l)] = 0, \end{aligned} \tag{A43}$$

$$y = \frac{k[1-P(l)-lQ(l)] - \frac{(b-kl)[1-P(l)-lQ(l)]}{b-k-l+1}}{(b-k)\left(\frac{1-P(l)-lQ(l)}{(b-k-l+1)Q(l)}\right)^l P(l)}, \tag{A44}$$

(A45)

$$y = \frac{(b-k-l+1)^{l-1}Q(l)^l\{k(b-k-l+1)[1-P(l)-lQ(l)]-(b-kl)[1-P(l)-lQ(l)]\}}{(b-k)[1-P(l)-lQ(l)]^l P(l)},$$

$$y = \frac{(b-k-l+1)^{l-1}Q(l)^l(bk-k^2-kl+k-b+kl)[1-P(l)-lQ(l)]}{(b-k)[1-P(l)-lQ(l)]^l P(l)}, \tag{A46}$$

$$y = \frac{(b-k-l+1)^{l-1}Q(l)^l(k-1)[1-P(l)-lQ(l)]}{[1-P(l)-lQ(l)]^l P(l)}, \tag{A47}$$

$$y = (k-1)\left(\frac{(b-k-l+1)Q(l)}{1-P(l)-lQ(l)}\right)^{l-1}\frac{Q(l)}{P(l)}. \tag{43}$$

Equation (44). To have $x > 1$ we need

$$\frac{1 - P(l) - lQ(l)}{(b - k - l + 1)Q(l)} > 1. \tag{A48}$$

For $b > k + l - 1$ we have

$$1 - P(l) - lQ(l) > (b - k - l + 1)Q(l), \tag{A49}$$

$$1 - P(l) > (b - k + 1)Q(l), \tag{A50}$$

$$b < k - 1 + \frac{1 - P(l)}{Q(l)}. \tag{A51}$$

So that the upper and lower bounds look more similar, we rewrite the upper bound on b by adding and subtracting l:

$$k + l - 1 < b < k + l - 1 + \frac{1 - P(l) - lQ(l)}{Q(l)}. \tag{44}$$

Suppose $b < k + l - 1$. Then from (A48) we have

$$1 - P(l) - lQ(l) < (b - k - l + 1)Q(l), \tag{A52}$$

$$1 - P(l) < (b - k + 1)Q(l), \tag{A53}$$

$$b > k - 1 + \frac{1 - P(l)}{Q(l)}, \tag{A54}$$

$$k + l - 1 > b > k - 1 + \frac{1 - P(l)}{Q(l)}. \tag{A55}$$

For this range to be nonempty, we need

$$k + l - 1 > k - 1 + \frac{1 - P(l)}{Q(l)}, \tag{A56}$$

$$0 > \frac{1 - P(l)}{Q(l)} - l, \tag{A57}$$

$$0 > \frac{1 - P(l) - lQ(l)}{Q(l)}, \tag{A58}$$

but this cannot be. We have $P(l) + lQ(l) = p^l + l(1 - p)p^{l-1}$, which are some of the terms in the binomial expansion of $[p + (1 - p)]^l = 1$. Since all of the terms are nonnegative (for $0 \geq p \geq 1$) the sum of some of the terms is no more than 1, so the right side of (A58) is nonnegative. Thus, the range is always empty.

Equation (45).

$$b < bP(l - 1) + l - 1 + \frac{1 - P(l) - lQ(l)}{Q(l)}, \tag{A59}$$

$$b[1 - P(l - 1)] < \frac{1 - P(l) - Q(l)}{Q(l)}, \tag{A60}$$

$$b[1 - P(l - 1)] < \frac{1 - P(l - 1)}{Q(l)}, \tag{A61}$$

$$b < \frac{1}{Q(l)}. \tag{A62}$$

Equation (46). To have $y < 1$ we need

$$(k-1)\left(\frac{(b-k-l+1)Q(l)}{1-P(l)-lQ(l)}\right)^{l-1}\frac{Q(l)}{P(l)} < 1. \tag{A63}$$

For $l \geq 2$

$$\frac{(b-k-l+1)Q(l)}{1-P(l)-lQ(l)}\left(\frac{(k-1)Q(l)}{P(l)}\right)^{1/(l-1)} < 1, \tag{A64}$$

$$b-k-l+1 \leq \frac{1-P(l)-lQ(l)}{Q(l)\left(\frac{(k-1)Q(l)}{P(l)}\right)^{1/(l-1)}}, \tag{A65}$$

$$b < k+l-1+\frac{1-P(l)-lQ(l)}{Q(l)}\left(\frac{P(l)}{(k-1)Q(l)}\right)^{1/(l-1)}. \tag{46}$$

The upper bound on b from (46) is greater than the lower bound from (44).

$$k+l-1+\frac{1-P(l)-lQ(l)}{Q(l)}\left(\frac{P(l)}{(k-1)Q(l)}\right)^{1/(l-1)} > k+l-1, \tag{A66}$$

$$\frac{1-P(l)-lQ(l)}{Q(l)}\left(\frac{P(l)}{(k-1)Q(l)}\right)^{1/(l-1)} > 0. \tag{A67}$$

All the terms on the left are positive for $0 < p < 1$.

Equation (47). We now consider when the upper bound on b from (46) is less than the upper bound from (44):

(A68)

$$k+l-1+\frac{1-P(l)-lQ(l)}{Q(l)}\left(\frac{P(l)}{(k-1)Q(l)}\right)^{1/(l-1)} < k+l-1+\frac{1-P(l)-lQ(l)}{Q(l)},$$

$$\frac{1-P(l)-lQ(l)}{Q(l)}\left(\frac{P(l)}{(k-1)Q(l)}\right)^{1/(l-1)} < \frac{1-P(l)-lQ(l)}{Q(l)}, \tag{A69}$$

$$0 < \frac{1-P(l)-lQ(l)}{Q(l)}\left[1-\left(\frac{P(l)}{(k-1)Q(l)}\right)^{1/(l-1)}\right]. \tag{A70}$$

As shown above (below (A58)), the first factor is always positive. For the second factor to be positive we need

$$1 > \left(\frac{P(l)}{(k-1)Q(l)}\right)^{1/(l-1)}, \tag{A71}$$

$$1 > \frac{P(l)}{(k-1)Q(l)}, \tag{A72}$$

$$(k-1)(1-p)p^{l-1} > p^l, \tag{A73}$$

$$(k-1)(1-p) > p, \tag{A74}$$

$$k-kp-1+p > p, \tag{A75}$$

$$k(1-p) > 1, \tag{A76}$$

$$k > \frac{1}{1-p}. \tag{47}$$

Equation (51). Equation (35) with $y = 1$ is

$$(b-k)P(l)x^l + (b-kl)Q(l)x - k[1-P(l)-lQ(l)] = 0. \tag{A77}$$

Let $x = 1+\delta$ with small δ and expand to second order. Let θ stand for quantities that approach 1 in the limit as δ approaches 0. In other words, θ is short for $[1 + o(1)]$, where δ is the variable that is approaching zero.

(A78)
$$(b-k)P(l)\left(1 + l\delta + \frac{l(l-1)\delta^2\theta}{2}\right) + (b-kl)Q(l)(1+\delta) = k[1-P(l)-lQ(l)],$$

$$\begin{aligned}&(b-k)P(l)\left(l\delta + \frac{l(l-1)\delta^2\theta}{2}\right) + (b-kl)Q(l)\delta \\ &\quad = k[1-P(l)-lQ(l)] - (b-k)P(l) - (b-kl)Q(l),\end{aligned} \tag{A79}$$

$$\delta = \frac{k[1-P(l)-lQ(l)] - (b-k)P(l) - (b-kl)Q(l)}{l(b-k)P(l)[1+(l-1)\theta\delta/2] + (b-kl)Q(l)}, \tag{A80}$$

$$\delta = \frac{k - bP(l) - bQ(l)}{b[lP(l)+Q(l)] - kl[P(l)+Q(l)] + (b-k)l(l-1)P(l)\theta\delta/2}, \tag{A81}$$

$$\delta = \frac{k - bP(l-1)}{b[lP(l)+Q(l)] - klP(l-1) + (b-k)l(l-1)P(l)\theta\delta/2}, \tag{A82}$$

$$\delta = \frac{k - bP(l-1)}{b[lP(l)+Q(l)] - klP(l-1)}\left(1 + \frac{(b-k)l(l-1)P(l)\theta\delta/2}{b[lP(l)+Q(l)] - klP(l-1)}\right)^{-1}, \tag{A83}$$

(51)
$$\delta = \frac{k - bP(l-1)}{b[lP(l)+Q(l)] - klP(l-1)}\left(1 + \frac{[k-bP(l-1)](b-k)l(l-1)P(l)\theta/2}{\{b[lP(l)+Q(l)] - klP(l-1)\}^2}\right)^{-1}.$$

Equation (53). Write (50) as

$$F_l \le e^X, \tag{A84}$$

with

$$X = \ln\{x^{-kl}[1 + (x^l-1)P(l) + l(x-1)Q(l)]^b\} \tag{A85}$$

$$= -kl\ln x + b\ln[1 + (x^l-1)P(l) + l(x-1)Q(l)]. \tag{A86}$$

Replace x with $1+\delta$:

$$X = -kl\ln(1+\delta) + b\ln\{1 + [(1+\delta)^l - 1]P(l) + lQ(l)\delta\}. \tag{A87}$$

Expanding X in a power series to second order gives

$$X = -kl\ln(1+\delta) + b\ln\left(1 + l\delta P(l) + \frac{l(l-1)P(l)\theta\delta^2}{2} + lQ(l)\delta\right)$$

$$= -kl\delta + \frac{kl\theta\delta^2}{2} + b\left(lP(l)\delta + \frac{l(l-1)\delta^2 P(l)\theta}{2} + lQ(l)\delta\right)$$

$$- \frac{b}{2}\left(lP(l)\delta + \frac{l(l-1)P(l)\theta\delta^2}{2} + lQ(l)\delta\right)^2\theta \tag{A88}$$

$$= -l[k - bP(l) - bQ(l)]\delta + \frac{kl + bl(l-1)P(l) - bl^2[P(l)+Q(l)]^2}{2}\theta\delta^2 \tag{A89}$$

$$= -l[k - bP(l-1)]\delta + \frac{kl + bl(l-1)P(l) - bl^2[P(l-1)]^2}{2}\theta\delta^2. \tag{A90}$$

Replace k by its definition in terms of α_3 (equation (52)) to obtain

$$X = -l[k - bP(l-1)]\delta + \frac{\{kl + bl(l-1)P(l) - bl^2[P(l-1)]^2\}\theta}{2}\delta^2 \tag{A91}$$

$$= -bl\alpha_3\delta + \frac{bl\{P(l-1) + \alpha_3 + (l-1)P(l) - l[P(l-1)]^2\}\theta}{2}\delta^2 \tag{A92}$$

$$= \left(-bl\alpha_3 + \frac{bl\{P(l-1) + \alpha_3 + (l-1)P(l) - l[P(l-1)]^2\}\theta\delta}{2}\right)\delta. \tag{A93}$$

Also replace k in (51) by its value in terms of α_3 to obtain

$$\delta = \frac{\alpha_3 b}{b[lP(l)+Q(l)] - b[P(l-1)+\alpha_3]lP(l-1)}$$

$$\times\left(1 + \frac{b\alpha_3\{b - b[P(l-1)+\alpha_3]\}l(l-1)P(l)\theta/2}{\{b[lP(l)+Q(l)] - b[P(l-1)+\alpha_3]l[P(l-1)]\}^2}\right)^{-1} \tag{A94}$$

$$= \frac{\alpha_3}{lP(l)+Q(l) - l[P(l-1)+\alpha_3]P(l-1)}$$

$$\times\left(1 + \frac{\alpha_3[1 - P(l-1) - \alpha_3]l(l-1)P(l)\theta/2}{\{lP(l)+Q(l) - l[P(l-1)+\alpha_3][P(l-1)]\}^2}\right)^{-1}. \tag{A95}$$

Since δ and α_3 go to zero together, this can be written as

$$\delta = \frac{\theta\alpha_3}{lP(l)+Q(l) - l[P(l-1)]^2}. \tag{A96}$$

Plugging the value of δ into the expression for X (equation (A93)) gives

$$X = \left(-bl\alpha_3 + \frac{bl\{P(l-1)+\alpha_3+(l-1)P(l)-l[P(l-1)]^2\}\theta\delta}{2}\right)\delta \tag{A97}$$

$$= \left(-bl\alpha_3 + \frac{bl\{P(l-1)+(l-1)P(l)-l[P(l-1)]^2+\alpha_3\}\theta\alpha_3}{2\{P(l-1)+(l-1)P(l)-l[P(l-1)]^2\}}\right)$$

$$\times \frac{\theta\alpha_3}{P(l-1)+(l-1)P(l)-l[P(l-1)]^2} \tag{A98}$$

$$= \left(-bl\alpha_3 + \frac{bl\theta\alpha_3}{2} + \frac{bl\theta\alpha_3^2}{2\{P(l-1)+(l-1)P(l)-l[P(l-1)]^2\}}\right)$$

$$\times \frac{\theta\alpha_3}{P(l-1)+(l-1)P(l)-l[P(l-1)]^2} \tag{A99}$$

$$= -\frac{bl\theta\alpha_3^2}{2\{P(l-1)+(l-1)P(l)-l[P(l-1)]^2\}}. \tag{A100}$$

Thus,

$$F_l \le e^{-bl\theta\alpha_3^2/(2\{P(l-1)+(l-1)P(l)-l[P(l-1)]^2\})}. \tag{53}$$

Equation (54). The derivation of (53) requires that $\delta = o(1)$. The step from (A98) to (A99) requires that α_3 be small compared to some other terms. Both conditions imply

$$\alpha_3 = \{lP(l)+Q(l)-l[P(l-1)]^2\}o(1). \tag{54}$$

Equation (56). By inclusion-exclusion, the sum for the region that defines R_k is equal to the sum over the entire area ($r_k(b,l,m,n,0)$), minus the sums over the various regions where a single j is required to be outside of R_k's region (l copies of $r_k(b,l,m,n,1)$), plus the sums over regions where two j's are required to be outside of R_k's region, etc.

Equation (58).

$$r_k(b,l,m,n,h) = \sum_{\substack{j_1<k\\ j_2<k\\ \cdots\\ j_h<k\\ j_{h+1},\ldots,j_l}} \binom{b}{j_1,j_2,\ldots,j_l,b-j_1-\cdots-j_l}$$

$$\times [Q(m)]^{j_1+\cdots+j_l}[1-P(m)-nQ(m)]^{b-j_1-\cdots-j_l} \tag{57}$$

$$= \sum_{\substack{j_1<k\\ j_2<k\\ \cdots\\ j_h<k\\ j_{h+1},\ldots,j_{l-1}}} \binom{b}{j_1,j_2,\ldots,j_{l-1},b-j_1-\cdots-j_{l-1}}$$

$$\times [Q(m)]^{j_1+\cdots+j_{l-1}}[1-P(m)-(n-1)Q(m)]^{b-j_1-\cdots-j_{l-1}} \tag{A101}$$

$$\cdots$$

$$= \sum_{\substack{j_1<k\\ j_2<k\\ \cdots\\ j_h<k}} \binom{b}{j_1,j_2,\ldots,j_h,b-j_1-\cdots-j_h}$$

$$\times [Q(m)]^{j_1+\cdots+j_h}[1-P(m)-(n-l+h)Q(m)]^{b-j_1-\cdots-j_h}. \tag{58}$$

Bound on first term of (61). For

$$\sum_{j_0<k} \binom{b}{j_0} [P(l)]^{j_0} [1-P(l)]^{b-j_0}$$

use the Chernoff bound from (23).

Equation (62).

$$\begin{aligned}
&\sum_{j_0<k} \binom{b}{j_0} [P(l)]^{j_0} \sum_{j<k-j_0} \binom{b-j_0}{j} [Q(l)]^j [1-P(l)-Q(l)]^{b-j_0-j} \\
&\quad \le x^{-k+1} y^{-k+1} \sum_{j_0,j} \binom{b}{j_0} [xyP(l)]^{j_0} \binom{b-j_0}{j} \\
&\quad\quad \times [xQ(l)]^j [1-P(l)-Q(l)]^{b-j_0-j} \qquad \text{(A102)} \\
&\quad \le x^{-k+1} y^{-k+1} [1+(xy-1)P(l)+(x-1)Q(l)]^b. \qquad \text{(A103)}
\end{aligned}$$

In real life, the Apriori Algorithm is used to analyze data that are more complex. Presumably, no one is interested in running the algorithm on truly random data. Rather, one is interested in the way in which the data differ from random. Nonetheless, we believe that analysis with this simple probability model brings out the main features of the performance of the algorithm. In principle, the techniques in this paper can be applied to more complex probability models of shopping. The challenge is to carry out the resulting calculations so that one can understand the implications of the formulas that result when the analysis is done on more general probability models.

With our probability model we calculate two quantities:

1. *Success rate*: the probability that a set is a success (i.e., passes the frequency test) and test.
2. *Failure rate*: the probability that a set is a candidate (i.e., passes the candidacy test) but not a success (i.e., fails the frequency test).

Notice that the success rate is a property of the probability model, not the algorithm. All correct algorithms will have the same success rate. The Apriori Algorithm never believes that an item set occurs k times without verifying the fact by counting occurrences in the data base. For algorithms that use this approach, the success rate represents unavoidable work. The Apriori Algorithm is clever in trying to reduce the failure rate. The failure rate represents work that one might hope to avoid.

Also, when there is a total of b baskets, b_A baskets with item A, and b_B items with item B, the Apriori Algorithm is not aware that there must be at least $2b - b_A - b_B$ baskets that contain both items A and B [11]. Similar ideas are explored in [7].

In the best case, the test for candidacy correctly predicts the result of the test for frequency, and the amount of time spent by the Apriori Algorithm is essentially the amount of time spent verifying that all the output sets should be output. A naive upper bound comes from the fact that each candidate is based on a previous success. Each success can lead to at most $|I|$ candidates. Many previous papers have focused on the total amount of work done by the Apriori Algorithm without concern as to the amount of output generated, but such studies can be misleading in that the amount of output is the main factor that determines how much work an Apriori-like algorithm will do, and the amount of output is a feature of the problem instance rather than of the algorithm. The simple lower and upper bound analyses say that the ratio between

the number of candidates and the number of successes (i.e, the output) is between 1 and $|I|$ (so long as there is at least one success).

Stronger worst-case bounds are given by Geerts, Goethals, and Van den Bussche [10]. They start by writing the number of successes on level l as the sum of distinct binomial coefficients where the first binomial coefficient has l as its bottom index and the largest possible top index. The remaining terms (if any) have smaller bottom indices. Their Theorem 1 says that the number of candidates on level $l+1$ is bounded by the number obtained from increasing each bottom index by 1. This bound is exact for some distributions, including one that results in no failures. Experiments show that in many cases this upper bound is close to the exact value.

At first sight, the approach of Theorem 1 of [10] looks quite different from the approach in this paper. However, in those cases where the number of candidates on level l is exactly $\binom{m_{l,l}}{l}$ for some $m_{l,l}$ there are just two differences. First, [10] considers only items that might still be active during the current level of the Apriori Algorithm ($m_{l,l}$). This is their main insight. Second, the upper limit corresponds to setting our p to 1. The work in [10] shows that the number of candidates on level l can never be more than $\binom{m_{l,l}}{l+1}$. If $m_{l,l}$ is much bigger than l, the ratio of candidates on level $l+1$ to candidates on level l is approximately $m_{l,l}/(l+1)$. When the number of candidates cannot be represented as an appropriate binomial, there are additional technical differences between the approach in [10] and in this paper, but they are not significant to a qualitative understanding of the situation.

It is not logically necessary that an algorithm verify occurrences by explicit counting. One alternative algorithm uses ideas that are the complement of those used by the Apriori Algorithm. The key idea for this complementary algorithm is that if some superset of a set J occurs at least k times, then so does set J. Also, if one changes the problem so that one needs only the maximal frequent item sets instead of all frequent item sets, then the amount of output needed is greatly reduced for some problem instances.

Acknowledgments. The authors want the express their thanks to Nele Dexters for helpful discussions and careful proofreading. They would also like to acknowledge the support of Indiana University Research and Technology Services for computing support during the experiments.

REFERENCES

[1] R. C. Agrawal, C. C. Aggarwal, and V. Prasad, *Depth first generation of long patterns*, in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, pp. 108–118.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, *Fast discovery of association rules*, in Advances in Knowledge Discovery and Data Mining, U. M. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy, eds., MIT Press, Cambridge, MA, 1996, pp. 307–328.

[3] R. Agrawal and R. Srikant, *Fast algorithms for mining association rules*, in Proceedings of the 1994 Very Large Data Bases Conference, Morgan–Kaufmann, 1994, pp. 487–499.

[4] R. Bayardo, *Efficiently mining long patterns from databases*, in Proceedings of the ACM SIGMOD International Conference of Management of Data, 1998, pp. 85–93.

[5] S. Brin, R. Motwani, and C. Silverstein, *Beyond market baskets: Generalizing association rules to correlations*, in Proceedings of the ACM SIGMOD International Conference of Management of Data, 1997, pp. 265–276.

[6] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, *Dynamic itemset counting and implication rules for market basket data*, in Proceedings of the ACM SIGMOD Conference on Management of Data, 1997, pp. 255–264.

[7] T. Calders, *Axiomatization and Deduction Rules for the Frequency of Itemsets*, Ph.D. Dissertation, University of Antwerp, 2003.
[8] H. Chernoff, *A measure of asymptotic efficiency for test of a hypothesis based on the sum of observations*, Ann. Math. Statist., 23 (1942), pp. 493–507.
[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.
[10] F. Geerts, B. Goethals, and J. Van den Bussche, *A tight upper bound on the number of candidate patterns*, in Proceedings of the 2001 IEEE International Conference on Data Mining, 2001, pp. 155–162.
[11] D. P. Groth and E. L. Robertson, *Discovering frequent itemsets in the presence of highly frequent items*, Rule Based Data Mining Workshop, in Conjunction with the 14th International Conference on Applications of Prolog, Springer-Verlag, New York, 2001, pp. 237–245.
[12] J. Han, J. Pei, and Y. Yin, *Mining frequent patterns without candidate generation*, in Proceedings of the ACM SIGMOD International Conference of Management of Data, 2000, pp. 1–12.
[13] IBM Quest Research Group, http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html#assocSynData
[14] L. V. S. Lakshmanan, R. T. Ng, J. Han, and A. Pang, *Optimization of constraint frequent set queries with* 2*-variable constraints*, in Proceedings of the ACM SIGMOD International Conference of Management of Data, 1999, pp. 157–168.
[15] M. Karr, *Summations in finite terms*, J. ACM, 28 (1981), pp. 305–350.
[16] H. Mannila, *Methods and problems in data mining*, in Proceedings of the International Conference on Database Theory, Springer-Verlag, New York, 1997, pp. 41–55.
[17] H. Mannila, H. Toivonen, and A. I. Verkamo, *Efficient algorithms for discovering association rules*, in Proceedings of the AAAI Workshop on Knowledge Discovery in Databases, 1994, pp. 181–192.
[18] R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang, *Exploratory mining and pruning optimizations of constrained association rules*, in Proceedings of the ACM SIGMOD International Conference of Management of Data, 1998, pp. 13–24.
[19] P. W. Purdom, Jr., and C. A. Brown, *The Analysis of Algorithms*, Holt, Rinehart and Winston, New York, 1985.
[20] S. Sarawagi, S. Thomas, and R. Agrawal, *Integrating association rule mining with relational database systems: Alternatives and implications*, in Proceedings of the ACM SIGMOD International Conference of Management of Data, 1998, pp. 343–354.
[21] H. Toivonen, *Discovery of Frequent Patterns in Large Data Collections*, Ph.D. Thesis, Report A-1996-5, University of Helsinki, 1996.
[22] S. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal, *Query flocks: A generalization of association-rule mining*, in Proceedings of the ACM SIGMOD International Conference of Management of Data, 1998, pp. 1–12.
[23] http://dataferrett.census.gov/TheDataWeb/index.html
[24] Z. Zheng, R. Kohavi, and L. Mason, *Real world performance of association rule algorithms*, in Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2001, pp. 401–406.

SIAM J. COMPUT.
Vol. 33, No. 6, pp. 1261–1279

FAST, DISTRIBUTED APPROXIMATION ALGORITHMS FOR POSITIVE LINEAR PROGRAMMING WITH APPLICATIONS TO FLOW CONTROL*

YAIR BARTAL†, JOHN W. BYERS‡, AND DANNY RAZ§

Abstract. We study combinatorial optimization problems in which a set of distributed agents must achieve a global objective using only local information. Papadimitriou and Yannakakis [*Proceedings of the 25th ACM Symposium on Theory of Computing*, 1993, pp. 121–129] initiated the study of such problems in a framework where distributed decision-makers must generate feasible solutions to *positive linear programs* with information only about local constraints. We extend their model by allowing these distributed decision-makers to perform local communication to acquire information over time and then explore the tradeoff between the amount of communication and the quality of the solution to the linear program that the decision-makers can obtain.

Our main result is a distributed algorithm that obtains a $(1+\epsilon)$ approximation to the optimal linear programming solution while using only a polylogarithmic number of rounds of local communication. This algorithm offers a significant improvement over the logarithmic approximation ratio previously obtained by Awerbuch and Azar [*Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, 1994, pp. 240–249] for this problem while providing a comparable running time. Our results apply directly to the application of network flow control, an application in which distributed routers must quickly choose how to allocate bandwidth to connections using only local information to achieve global objectives. The sequential version of our algorithm is faster and considerably simpler than the best known approximation algorithms capable of achieving a $(1+\epsilon)$ approximation ratio for positive linear programming.

Key words. linear programming, approximation algorithm, primal-dual, flow control

AMS subject classifications. 68W15, 68W25

DOI. 10.1137/S0097539700379383

1. Introduction.

Processors in a distributed environment must make decisions based only on local data; thus fast distributed algorithms must often do without global information about the system as a whole. This is exactly why computing many target functions in distributed models quickly is provably hard [15]. However, quite surprisingly, some of the most interesting global optimization problems can be very closely approximated based only on local information and a modest amount of local communication.

Our work is motivated by the application of developing flow control policies which must achieve global objective functions. Flow control is the mechanism by which routers of a network distribute the available network bandwidth across connections. In our work, routing policies determine the routes in the network that connections must use to transmit packets. Regulating the rates at which the connections may inject data along these fixed routes is the problem of flow control. This connection-oriented, or rate-based, approach to flow control is a standard for routing available bit-rate

*Received by the editors October 11, 2000; accepted for publication (in revised form) December 30, 2003; published electronically August 6, 2004. Much of this work was completed while the authors were affiliated with the International Computer Science Institute in Berkeley, CA, and the University of California, Berkeley. This research was supported in part by NSF operating grants CCR-9304722 and NCR-9416101 and by a grant from the U.S.-Israel Binational Science Foundation. The work presented in this paper appeared in preliminary form in [6] and [10].

http://www.siam.org/journals/sicomp/33-6/37938.html

†Department of Computer Science, Hebrew University, Jerusalem, Israel (yairb@cs.huji.ac.il).

‡Department of Computer Science, Boston University, Boston, MA (byers@cs.bu.edu).

§Department of Computer Science, Technion, Haifa, Israel (danny@cs.technion.ac.il).

traffic in ATM networks [9] and is expected to become widely used in packet-switched networks. In this approach, each router in the network must make regulatory decisions based only on local information, which typically consists of the current transmission rates of connections using the router. Most existing flow control policies try to satisfy local objective functions such as *max-min fairness* [13, 7, 1, 11]. However, there are many other practical scenarios in which global objective functions are the appropriate choice. For example, in a commercial intranetwork in which users are paying for use of the network bandwidth (possibly at different rates), the administrator would want to use a flow control policy which maximizes total revenue. We express such a flow control policy objective as a *positive linear program*, a linear program (LP) in which all entries of the constraint matrix are nonnegative. Complicating the issue is the problem that routers must generate feasible solutions to this LP quickly, and based only on available information.

Motivated by this application and other related applications, Papadimitriou and Yannakakis considered the problem of having distributed decision-makers assign values to a set of variables in an LP, where the agents have limited information [18]. In one scenario that they describe, each agent, acting in isolation, must set the value of a single primal variable, knowing only the constraints affecting that variable in the LP. In the context of flow control, where the objective is to maximize the total flow through the network, this corresponds to a setting in which connections only know how many other connections share each of the routers they intend to use. When all edge capacities are 1, their "safe" algorithm sets each connection's flow to the reciprocal of the maximum number of connections which share an edge with that connection. It is not hard to see that the worst-case approximation ratio achieved by the "safe" algorithm is $\Theta(\Delta)$, where Δ is the maximum number of connections that share an edge. They also prove that the "safe" algorithm achieves the best possible worst-case ratio when agents may not communicate, leaving open the possibility that much better ratios can be obtained when agents can interact.

We extend their model to allow computation to proceed in a sequence of *rounds*, in each of which agents can communicate a fixed-size message to their immediate neighbors, where agents are neighbors if and only if they share one or more constraints in the LP. Our goal is to determine the number of rounds necessary to achieve a $(1 + \epsilon)$ approximation ratio to the optimum LP solution. Although we focus on the application of flow control, this study could also be performed on a range of resource allocation problems, including those described in [18]. We note that similar models for describing the interaction between connections and routers in both theoretical and practical evaluations of flow control policies have been suggested in [3, 1, 5, 16].

Of course, a centralized administrator with complete information could solve the problem exactly using one of the well-known polynomial-time algorithms for linear programming (see, for example, [14]). Recently, however, much faster algorithms that produce approximate solutions to positive LPs to within a $(1 + \epsilon)$ factor of optimal have been designed. The sequential algorithm of Plotkin, Shmoys, and Tardos [19] repeatedly identifies a globally minimum weight path and pushes more flow along that path. More recently, faster approximation algorithms have been considered for several related flow and bin-packing problems [8, 12] using this same principle of repeatedly choosing a good flow, and incrementally increasing the rate of that flow. The technical difficulty is to balance the amount of flow increase, such that the required approximation is achieved, with the number of needed steps (i.e., running time). In all of these algorithms a global operator choosing the appropriate unsatisfied constraints is used. Moreover, the more general multicommodity flow problem cannot be formulated as

a positive LP, unless the number of dual variables or the number of constraints is exponential. In these cases one must use a separation oracle, since only polynomially many flows can be handled. For positive LPs, however, the number of constraints is polynomially bounded and thus one can increase all dual variables simultaneously. This is the approach taken by the algorithm of Luby and Nisan [17]. This algorithm, which has both a fast sequential and parallel implementation, repeatedly performs a global median selection algorithm on the values of the dual variables to choose a threshold, then increases values of dual variables above this threshold. Although these algorithms have efficient implementations, they both perform global operations, which makes them unsuitable for fast distributed implementation, since emulating those global operations requires a polynomial number of distributed rounds, and we are interested in much more time-efficient solutions.

The only previously known result for a distributed flow control algorithm with a global objective function is an algorithm of Awerbuch and Azar [3] which gives a logarithmic approximation ratio and also runs in a polylogarithmic number of rounds. Their algorithm is based on fundamental results from competitive analysis [2, 4]. The deterministic algorithm we present produces $(1+\epsilon)$ approximate solutions to positive LPs, both in general and for the flow control problem, and builds on ideas used in these other algorithms [3, 17, 19]. Our algorithm is most closely related to the algorithm of Luby and Nisan but eliminates the need for the complex global selection operations and a global normalization step upon termination, enabling fast implementation in a distributed setting. Those simplifications carry over to serial and parallel settings as well, where we have a dramatically simpler implementation which saves a $\frac{1}{\epsilon}$ factor in the running time over the Luby–Nisan algorithm. Finally, we can parameterize the algorithm to quantify a tradeoff between the number of rounds and the quality of the approximation. In practice, we can run the algorithm for any number of *phases*, with the guarantee that after a constant number of phases, we have a logarithmic factor approximation, and after a logarithmic number of phases, we have a $(1+\epsilon)$ approximation.

The rest of the paper is organized as follows. We begin with a presentation of our algorithm first as an easily understandable and implementable serial algorithm for approximately solving positive linear programming in section 2. In section 2.3, we prove that the algorithm achieves a $(1+\epsilon)$ approximation ratio, and we analyze its running time. We formulate our distributed model and give an explanation of the correspondence between flow control policies and positive LPs in section 3. Then in section 3.2, we present the distributed implementation applicable to the flow control problem and explain the modifications to the analysis for this case.

2. Sequential approximation algorithms. We consider positive LPs represented in the following standard form, which is well known to be as general as arbitrary positive linear programming.

PRIMAL	DUAL
$\max Y = \sum_{j=1}^{n} y_j$	$\min X = \sum_{i=1}^{m} x_i$
$\forall i, \sum_j a_{ij} y_j \leq 1$	$\forall j, \sum_i a_{ij} x_i \geq 1$
$\forall j,\ y_j \geq 0$	$\forall i,\ x_i \geq 0$
$\forall i, j,\ a_{ij} \geq 0$	$\forall i, j,\ a_{ij} \geq 0$

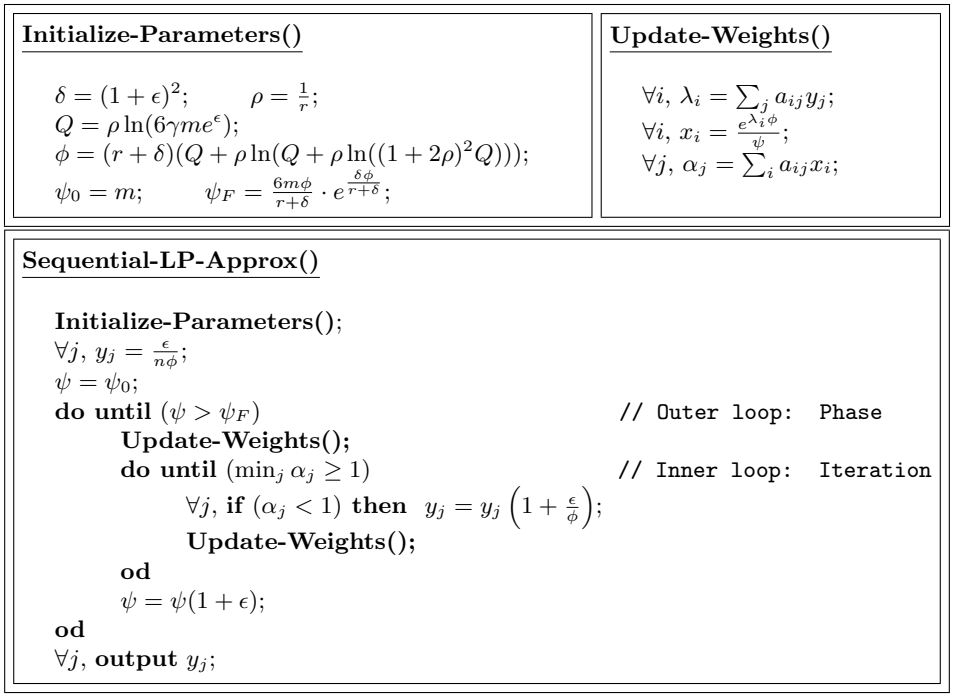


FIG. 1. *The sequential positive LP approximation algorithm.*

We will further assume that the LP is presented to the algorithm in a normalized form in which the a_{ij} are either 0 or satisfy $\frac{1}{\gamma} \leq a_{ij} \leq 1$. In the sequential case, one can convert an arbitrary LP in standard form into an LP in normalized form in linear time simply by dividing all constraints by $a_{max} = \max a_{ij}$ and setting $\gamma = \frac{a_{max}}{a_{min}}$ (where $a_{min} = \min\{a_{ij} | a_{ij} > 0\}$). We will describe how to perform this transformation efficiently in a distributed setting in section 3.

2.1. Overview of results. The parameterized algorithm for approximately solving a positive LP in normalized form is given in Figure 1. The main theorem that we will prove about the performance of this algorithm relates the quality of the approximation to the running time as follows.

THEOREM 1. *For any $0 < \epsilon \leq 1$ and $0 < r \leq \ln(\gamma m)$, the algorithm produces a feasible $(r + (1+\epsilon)^2)$ approximation to the optimum primal linear programming solution and runs in $O\left(\frac{nm \ln(\frac{\gamma m}{r})}{r\epsilon}\right)$ time.*

The following corollary clarifies the tradeoff between the running time and the quality of the approximation and follows directly from Theorem 1.

COROLLARY 2. *For any $\epsilon \leq 1$, the algorithm produces a $(1+\epsilon)$ approximation to the optimum in $O\left(\frac{nm \ln(\frac{\gamma m}{\epsilon})}{\epsilon^2}\right)$ time. For any $1 \leq r' \leq \ln(\gamma m)$, the algorithm produces a $(1+r')$ approximation to the optimum in $O\left(\frac{nm \ln^2(\gamma m)}{r'}\right)$ time.*

Proof of Corollary 2. To prove the first claim of the corollary, set $r = \epsilon$ in Theorem 1, and to prove the second, set $r = 1 + r' - (1+\epsilon)^2$ and choose ϵ so that $0 < \epsilon < \sqrt{2} - 1$, which implies $r > 0$ in Theorem 1. □

2.2. Description of the algorithm. In the sequential implementation of our algorithm presented in Figure 1, the main body of the program in the bottom panel

runs in a sequence of *phases*, the number of which ultimately depends only on the desired approximation ratio. Within a phase, values of appropriate primal variables y_j are increased monotonically until a threshold is reached. We refer to a set of increase operations across all primal variables constituting the inner loop of our main program as an *iteration* of our algorithm, noting that the number of iterations per phase may vary. We will demonstrate that at the time slot t ending each phase, the nonnegative settings for the primal variables $y_j(t)$ and the dual variables $x_i(t)$ are primal and dual feasible, respectively, satisfying the constraints below.

$$\textbf{Primal Feasibility: } \forall i, \sum_{j=1}^{n} a_{ij} y_j(t) \leq 1.$$

$$\textbf{Dual Feasibility: } \forall j, \sum_{i=1}^{m} a_{ij} x_i(t) \geq 1.$$

2.2.1. Moving between pairs of feasible solutions. Since the values of primal variables increase monotonically in our algorithm, it is crucial to carefully select the variables to increase in each phase. To this end, our algorithm uses exponential weight functions which have been employed in a variety of related contexts, including [2, 4, 17, 19, 3]. To do so, we associate each dual constraint with a measure $\alpha_j(t) = \sum_i a_{ij} x_i(t)$, and we associate each primal constraint with a measure $\lambda_i(t) = \sum_j a_{ij} y_j(t)$. Throughout the algorithm, the values of the dual variables x_i are tied to the values of "neighboring" primal variables y_j by the exponential weighting function defined in **Update-Weights()**:

$$\forall t : x_i(t) = \frac{e^{\lambda_i(t)\phi}}{\psi} = \frac{e^{\Sigma_j a_{ij} y_j(t)\phi}}{\psi},$$

where ϕ is a constant which again depends only on the desired approximation ratio, and ψ is a scaling factor which is initialized to m, then grows geometrically with the phase number until it reaches ψ_F, which is the termination condition.

By establishing this connection between primal and dual variables, when the value $\alpha_j(t)$ is less than 1, the dual constraint $\sum_i a_{ij} x_i(t) \geq 1$ is violated, but can be satisfied by a sufficient increase in y_j. This relationship suggests the following idea for an algorithm, which we employ. Start with a pair of dual and primal feasible solutions. Scale down the dual feasible solution by a multiplicative factor, making the solution dual infeasible, and causing some dual constraints to be violated. Then move back to a dual feasible solution by increasing those primal variables j for which $\alpha_j(t) < 1$, and repeat the process until a satisfactory approximation is achieved.

A hypothetical depiction of the intermediate primal and dual feasible solutions $Y(t) = \sum_j y_j(t)$ and $X = \sum_i x_i(t)$ at the end of each phase relative to the value of the optimal solution is shown in Figure 2. We maintain the invariant that the values of the intermediate primal feasible solutions are monotonically nondecreasing over time, but no such guarantee is provided for the intermediate dual feasible solutions. Linear programming feasibility ensures that values of intermediate primal feasible solutions are necessarily smaller than or equal to the value of the program, denoted by OPT, and similarly, values of intermediate dual feasible solutions are necessarily larger than or equal to the value of the program. Upon termination, we prove that the final (and maximal) primal feasible solution is within a desired (in this case, $1 + \epsilon$) factor of the minimal intermediate dual feasible solution. By linear programming duality, this

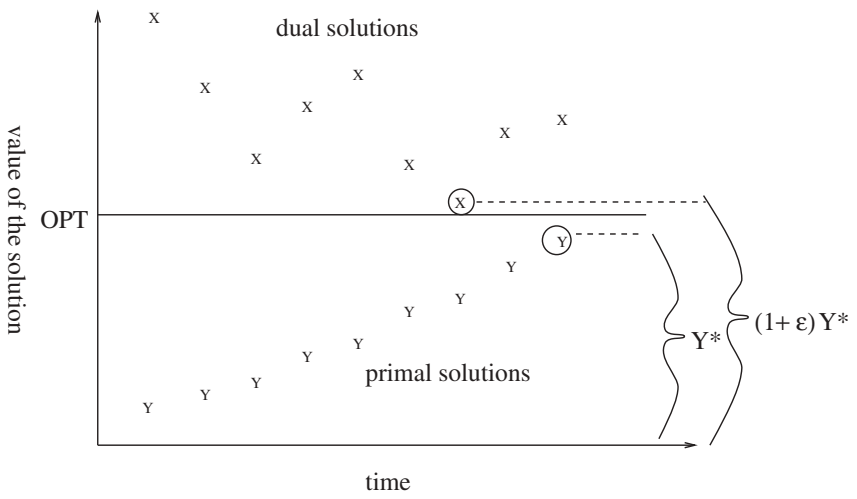


FIG. 2. *Intermediate primal and dual feasible solutions.*

implies that the final primal feasible solution gives the desired approximation to the value of the program.

In the sequential implementation presented in Figure 1, the bottleneck operation is to recompute the α_j's after each iteration, which takes $O(nm)$ time. In fact, the running time of this bottleneck operation can be more precisely written as $O(E)$, where E is the total number of nonzero entries of the constraint matrix of the LP. When multiplied by the total number of iterations, which we demonstrate to be at most polylogarithmic in section 2.3, we have the bound on the total sequential running time.

2.3. Analysis of the algorithm. In this section, we bound the approximation ratio of our approximation algorithm and present an analysis of its sequential running time. We will later extend both of these results in a straightforward way to the distributed case.

First we prove a claim we made earlier, that at the end of each phase both the primal and the dual solutions are feasible. From the definition of the algorithm, the values of primal variables increase monotonically, and therefore the values of intermediate primal solutions also increase monotonically. Thus to carry out the analysis of the approximation ratio, it will remain only to prove that the value of the final primal feasible solution and the value of the minimal dual feasible solution are at most a $(1+\epsilon)$ factor apart. In the proof we use the following three facts, which all follow from the initialization of the parameters given in **Initialize-Parameters()** in Figure 1.

FACT 3. $\phi \geq \epsilon$.

Proof. Fact 3 immediately follows from the definition of ϕ and from the fact that $\epsilon \leq 1$. □

FACT 4. $\phi = O\left(\frac{(r+\delta)\ln\left(\frac{\gamma m}{r}\right)}{r}\right)$.

Proof. By definition, $\phi = (r+\delta)(Q + \rho\ln(Q + \rho\ln((1+2\rho)^2 Q)))$ and $Q = \rho\ln(6\gamma m e^{\epsilon})$. Thus $Q \geq \rho$ and we have $\phi = O((r+\delta)(Q+\rho\ln Q))$. Since by definition

we also have $Q = O(\rho \ln(\gamma m))$ (recall that ρ is a shorthand for $1/r$), it suffices to show

$$\rho \ln(\gamma m) + \rho \ln(\rho \ln(\gamma m)) = O(\rho \ln(\rho \gamma m)).$$

Now since $r \leq \ln(\gamma m)$, we have that $\ln(\gamma m) = O(\ln(\rho \gamma m))$, and $\ln(\rho(\ln(\gamma m)) = O(\ln \rho + \ln \ln(\gamma m)) = O(\ln \rho + \ln(\gamma m)) = O(\ln(\rho \gamma m))$, and the result follows directly. ∎

FACT 5. $e^{\phi - \epsilon} \geq \gamma \psi$.

Proof. Substituting ψ_F for ψ and multiplying by e^{ϵ}, we must show

$$e^{\phi} \geq (\gamma e^{\epsilon}) \frac{6m\phi}{r+\delta} e^{\frac{\delta \phi}{r+\delta}}$$

or

$$e^{\frac{r\phi}{r+\delta}} \geq (6\gamma m e^{\epsilon}) \left(\frac{\phi}{r+\delta} \right)$$

or

$$e^{\frac{\phi}{r+\delta}} \geq \left(6\gamma m e^{\epsilon} \frac{\phi}{r+\delta} \right)^{\frac{1}{r}}.$$

Now by taking the natural logarithm of both sides above, we need to show

$$\frac{\phi}{r+\delta} \geq \frac{1}{r} \left[\ln(6\gamma m e^{\epsilon}) + \ln \left(\frac{\phi}{r+\delta} \right) \right].$$

Recall from the definitions that $\rho = \frac{1}{r}$, $Q = \rho \ln(6\gamma m e^{\epsilon})$, and $\frac{\phi}{r+\delta} = Q + \rho \ln(Q+P)$, where $P = \rho \ln((1+2\rho)^2 Q)$. Therefore, it is enough to prove the following:

$$\begin{aligned} &Q + \rho \ln(Q+P) - Q - \rho \ln(Q + \rho \ln(Q+P)) \\ &\quad = \rho \ln \left(\frac{Q+P}{Q + \rho \ln(Q+P)} \right) \geq 0. \end{aligned}$$

Since P and ρ are clearly nonnegative, and $Q \geq 1$, to show that $\rho \ln\left(\frac{Q+P}{Q+\rho \ln(Q+P)}\right)$ is nonnegative we need $P \geq \rho \ln(Q+P)$ or, substituting for P,

$$\begin{aligned} \rho \ln((1+2\rho)^2 Q) &\geq \rho \ln(Q + \rho \ln((1+2\rho)^2 Q)), \\ (1+2\rho)^2 Q &\geq Q + \rho \ln((1+2\rho)^2 Q), \\ 2Q(2\rho+2) &\geq \ln((1+2\rho)^2 Q). \end{aligned}$$

Using $2\ln((1+2\rho)Q) \geq \ln((1+2\rho)^2 Q)$, and $2Q(2\rho+2) \geq 2Q(2\rho+1)$, we have to show

$$2Q(2\rho+1) \geq 2\ln((1+2\rho)Q).$$

The final inequality follows from the fact that $x \geq \ln x$, which completes the proof. ∎

2.4. Feasibility. Recall that the algorithm maintains the invariant that dual feasibility is achieved prior to each increase in ψ.

FACT 6 (dual feasibility). *At the end of each phase, for all j, $\alpha_j \geq 1$.*

We next prove that the y_j's are primal feasible throughout the execution of the algorithm, using Claim 7 to help perform the induction. In the proof it will be convenient to treat the initial increase of y_j's from 0 to their initialized value as iteration 0.

CLAIM 7. *For all i and for every iteration, $\Delta\lambda_i \leq \frac{\epsilon}{\phi}$.*

CLAIM 8 (primal feasibility). *For all i, $\lambda_i \leq 1$ throughout the execution of the algorithm.*

We prove these two claims simultaneously by induction over iterations of the algorithm.

Proof. Let $J_i = \{j | a_{ij} > 0\}$. The first step is to prove that the claims hold for iteration 0:

$$\lambda_i = \sum_{j \in J_i} a_{ij} y_j \leq \sum_{j \in J_i} a_{ij} \frac{\epsilon}{n\phi} \leq \frac{\epsilon}{\phi}.$$

Since $\phi \geq \epsilon$ this also implies that Claim 8 holds at iteration 0.

Consider a subsequent iteration, and let Δv denote the change in variable v during that iteration. We have that for all j, $\Delta y_j \leq y_j \frac{\epsilon}{\phi}$ by the rate of increase in an iteration, so for all i,

$$\Delta\lambda_i = \sum_j a_{ij} \Delta y_j \leq \sum_j a_{ij} y_j \frac{\epsilon}{\phi} = \lambda_i \frac{\epsilon}{\phi} \leq \frac{\epsilon}{\phi},$$

where the final inequality holds by the inductive hypothesis of Claim 8. This completes the proof of Claim 7.

To complete the proof of Claim 8, we consider two cases for λ_i separately. We first consider values i for which $\lambda_i < 1 - \frac{\epsilon}{\phi}$ prior to an iteration. From the proof of Claim 7 we have that after such an iteration, $\lambda_i' \leq \lambda_i + \frac{\epsilon}{\phi} < 1$, giving the desired result.

Next we consider those i for which $\lambda_i \geq 1 - \frac{\epsilon}{\phi}$ prior to an iteration. Fix such an i and fix any $j \in J_i$. We have that

$$\alpha_j = \sum_k a_{kj} x_k \geq a_{ij} x_i = a_{ij} \frac{e^{\lambda_i \phi}}{\psi} \geq a_{ij} \frac{e^{\phi - \epsilon}}{\psi}.$$

By Fact 5, we have that by our choice of ϕ, $e^{\phi-\epsilon} \geq \gamma\psi$, and hence $\alpha_j \geq a_{ij}\gamma \geq 1$, by the definition of γ. By the increase rule in the algorithm, we never increase the value of primal variable y_j if $\alpha_j \geq 1$, so in fact no primal variable in J_i increases in this iteration. Therefore, λ_i does not increase during this iteration and remains smaller than 1 by the induction hypothesis, completing the proof. ∎

2.5. Proof of a $(1+\epsilon)$ approximation ratio. We now turn to bound the approximation ratio obtained by the algorithm stated as the first half of Theorem 1.

CLAIM 9. *For any $0 < \epsilon \leq 1$ and $0 < r \leq \ln(\gamma m)$, the algorithm produces a feasible $(r + (1+\epsilon)^2)$ approximation to the optimum primal linear programming solution.*

We use the notation $\Delta Y = \sum_j \Delta y_j$ to denote the aggregate change in the y values over the course of an iteration and similar notation for other variables. We begin with the following lemma.

LEMMA 10. *For every iteration,*

$$\frac{\Delta X}{\Delta Y} \leq \phi(1+\epsilon).$$

Proof. As $\epsilon \leq 1$, we have from Claim 7 that $\Delta\lambda_i\phi \leq \epsilon \leq 1$. It follows that

$$\begin{aligned}\Delta x_i &= x_i\left(e^{\Delta\lambda_i\phi} - 1\right)\\ &\leq x_i\Delta\lambda_i\phi(1+\Delta\lambda_i\phi)\\ &\leq x_i\Delta\lambda_i\phi(1+\epsilon),\end{aligned}$$

using the inequality $e^z - 1 \leq z(1+z)$ for $z \leq 1$.

Prior to a given iteration, let $S = \{j|\alpha_j < 1\}$. S is the set of indices of primal variables which will be active in the upcoming iteration, i.e., those variables y_j whose values will increase in the iteration. (Recall that a variable y_j may be increased several times in a phase.) The lemma follows from the following sequence of inequalities:

$$\begin{aligned}\Delta X &= \sum_i \Delta x_i \leq \sum_i x_i\Delta\lambda_i\phi(1+\epsilon)\\ &= \sum_i x_i \sum_{j\in S} a_{ij}\Delta y_j\phi(1+\epsilon)\\ &= \sum_{j\in S}\Delta y_j \sum_i a_{ij}x_i\phi(1+\epsilon)\\ &= \sum_{j\in S}\Delta y_j\alpha_j\phi(1+\epsilon)\\ &< \Delta Y\phi(1+\epsilon).\end{aligned}$$

The final inequality holds from the definition of S. □

In stating and proving the next lemma, we require a more precise description of our notation. We now consider the change in the values of the dual variables X over the course of a *phase*. In the proof, we let X' denote the sum of the x_i at the end of the current phase, and we let X denote the sum of the x_i at the end of the previous phase, i.e., before they are scaled down. We let ΔX denote the change in the sum of the x_i over the course of the current phase. We further define X^* to be the minimum over all dual feasible solutions obtained at the end of each phase and let Y_L be the primal feasible solution obtained at the end of the final phase. Fact 6 and Claim 8, respectively, imply that X^* is dual feasible and Y_L is primal feasible. In conjunction with Lemma 11 below, this implies the approximation result stated in Claim 9, by linear programming duality.

LEMMA 11.

$$Y_L \geq \frac{X^*}{r + (1+\epsilon)^2}.$$

Proof. We prove the lemma for two separate cases: the first is an easy case in which the initial primal feasible solution is a close approximation to the optimum, and in the second we repeatedly apply Lemma 9 to bound the ratio between X^* and Y_L. Let X_0 denote the value for X before the initialization of the y_j's, that is, $X_0 = \sum_i \frac{e^0}{m} = 1$, and let X_1 denote the value for X after the first phase. Similarly, let X_L denote the value of the dual solution at the end of the final phase.

Case I. $X_L \leq \frac{r+\delta}{\phi(1+\epsilon)}(X_1 - X_0)$. Letting Y_1 denote the value of the primal solution at the end of the first phase, we apply Lemma 9 to the iterations comprising the first phase, giving us

$$X_1 - X_0 \leq Y_1\phi(1+\epsilon).$$

Since the values of the primal feasible solutions increase monotonically throughout the course of the algorithm, the lemma holds by the following sequence of inequalities:

$$X^* \leq X_L \leq (r+\delta)Y_1 \leq (r+(1+\epsilon)^2)Y_L.$$

Case II. $X_L > \frac{r+\delta}{\phi(1+\epsilon)}(X_1 - X_0)$. Since the values X are scaled by a $\frac{\psi}{\psi'} = \frac{1}{1+\epsilon}$ factor just following the end of each phase, the earlier definitions imply that

$$X' = X\frac{\psi}{\psi'} + \Delta X.$$

By rewriting this expression and applying the inequality $e^z \geq 1 + z$, we have

$$X' = X\frac{\psi}{\psi'}\left(1 + \frac{\Delta X(1+\epsilon)}{X}\right) \leq X\frac{\psi}{\psi'}\left(e^{\frac{\Delta X(1+\epsilon)}{X}}\right).$$

Now, using $X^* \leq X$ and applying Lemma 10 yields

$$X' \leq X\frac{\psi}{\psi'}\left(e^{\frac{\Delta Y\phi(1+\epsilon)^2}{X^*}}\right).$$

Let ψ_1 and ψ_L to be the initial value of ψ and the value of ψ in the last phase of the algorithm, i.e., $\psi_L < \psi_F$. Using the bound above repeatedly to compare X_L with X_1 gives us

$$X_L \leq X_1\frac{\psi_1}{\psi_L}\left(e^{\frac{Y_L\phi(1+\epsilon)^2}{X^*}}\right). \tag{1}$$

Since X_L is dual feasible and the optimal solution is bounded below by 1 (by the normalized form of the program) we have that $X_L \geq 1 = X_0$. Also note that by the assumption $r \leq \ln(\gamma m)$, we have $\phi \geq (r+\delta)$. We can now use these facts in conjunction with the fact that $X_L > \frac{r+\delta}{\phi(1+\epsilon)}(X_1 - X_0)$ to obtain

$$X_1 < \frac{X_L\phi(1+\epsilon)}{r+\delta} \leq X_L\left(\frac{\phi(1+\epsilon)}{r+\delta} + 1\right) \leq X_L\frac{\phi(2+\epsilon)}{r+\delta}.$$

Using the bound above in (1) and observing that $\psi_1 = m$ and $\psi_L \geq \psi_F/(1+\epsilon)$ we get

$$e^{\frac{Y_L\phi(1+\epsilon)^2}{X^*}} \geq \frac{\psi_F}{m\frac{\phi(2+\epsilon)}{r+\delta}(1+\epsilon)} = \frac{6m\frac{\phi}{r+\delta}e^{\frac{\delta\phi}{r+\delta}}}{m(2+\epsilon)(1+\epsilon)\frac{\phi}{r+\delta}} = \frac{6e^{\frac{\delta\phi}{r+\delta}}}{(2+\epsilon)(1+\epsilon)} \geq e^{\frac{(1+\epsilon)^2\phi}{r+(1+\epsilon)^2}},$$

where the final inequality holds by substituting $\delta = (1+\epsilon)^2$ and using $\epsilon \leq 1$. We therefore get

$$\frac{Y_L}{X^*} \geq \frac{1}{r+(1+\epsilon)^2},$$

concluding the proof of the lemma for Case II. □

2.6. Running time. For the algorithm provided so far, we have the following running time bounds, which are slightly weaker than those stated in Theorem 1. These weaker bounds are presented since a natural translation of these time bounds and the preceding analysis extends directly to the distributed algorithm which we present in section 3. After proving these bounds, we provide a simple improvement which applies only in the sequential case and which is used to give the time bounds stated in Theorem 1.

CLAIM 12. *The sequential algorithm runs in* $O\left(\frac{(r+1)nm\ln^2\left(\frac{\gamma m}{r}\right)\ln\left(\frac{(r+1)\gamma n\ln m}{r\epsilon}\right)}{r^2\epsilon^2}\right)$ *time.*

Proof. We bound the number of phases by measuring the change in ψ, which increases by a $(1+\epsilon)$ factor per phase. From the definitions in Figure 1, and using Fact 4 for the final equality, we have that

$$\left\lceil \log_{1+\epsilon}\left(\frac{\psi_F}{\psi_0}\right)\right\rceil = O\left(\frac{\frac{\delta\phi}{r+\delta}+\ln\left(\frac{\phi}{r+\delta}\right)}{\epsilon}\right) = O\left(\frac{\phi}{(r+\delta)\epsilon}\right) = O\left(\frac{\ln(\frac{\gamma m}{r})}{r\epsilon}\right).$$

We now bound the number of iterations in a phase by computing the maximum number of iterations needed to increase all α_j values above 1. For a given j, we say that y_j is *large* once $y_j \geq \frac{\gamma}{\phi}\ln(\gamma\psi)$. We show that once y_j is large, $\alpha_j \geq 1$, and therefore j no longer participates in the phase. Let the set $I_j = \{i | a_{ij} > 0\}$. Therefore for all $i \in I_j$,

$$\lambda_i = \sum_k a_{ik}y_k \geq \frac{1}{\gamma}y_j \geq \frac{1}{\phi}\ln(\gamma\psi),$$

$$\alpha_j = \sum_i a_{ij}x_i = \sum_i a_{ij}\cdot\frac{e^{\lambda_i\phi}}{\psi} \geq 1.$$

Initially, $y_j = \frac{\epsilon}{n\phi}$ and at every iteration it increases by a factor of $1+\frac{\epsilon}{\phi}$. Therefore the number of iterations y_j can participate in before y_j becomes large is at most

$$\left\lceil \log_{1+\frac{\epsilon}{\phi}}\left(\frac{n\phi}{\epsilon}\cdot\frac{\gamma}{\phi}\ln(\gamma\psi)\right)\right\rceil = O\left(\frac{\phi}{\epsilon}\cdot\ln\left(\frac{\gamma n}{\epsilon}\ln(\gamma\psi)\right)\right).$$

From Fact 5 we have $\ln(\gamma\psi) = O(\phi)$. By using this fact, and by another application of Fact 4, we bound the number of iterations during a phase by

$$O\left(\frac{\phi}{\epsilon}\cdot\ln\left(\frac{\gamma n\phi}{\epsilon}\right)\right) = O\left(\frac{r+\delta}{r\epsilon}\ln\left(\frac{\gamma m}{r}\right)\ln\left(\frac{\gamma n}{\epsilon}\frac{r+\delta}{r}\ln\left(\frac{\gamma m}{r}\right)\right)\right)$$

$$= O\left(\frac{(r+1)\ln\left(\frac{\gamma m}{r}\right)\ln\left(\frac{(r+1)\gamma n\ln m}{r\epsilon}\right)}{r\epsilon}\right).$$

Note that the term $\frac{r+1}{r}$ cannot be removed since r can be any value satisfying $0 < r \leq \ln(\gamma m)$. Multiplying this by the earlier bound on the number of phases and using the fact that each iteration can be computed in $O(nm)$ time, this completes the proof of Claim 12. ∎

To prove the time bound stated in the second half of Theorem 1, we need to give a sequential algorithm which completes in $O\left(\frac{nm\ln(\frac{\gamma m}{r})}{r\epsilon}\right)$ time. This running time can be achieved by an algorithm which performs exactly one iteration per phase. In

the sequential case, we accomplish this by increasing a candidate y_j not merely by a multiplicative factor of $1 + \frac{\epsilon}{\phi}$ per iteration, but by increasing it by the amount which causes α_j to reach a value of 1 directly. (Note that this procedure is straightforward to implement in the sequential case, but not in the distributed case.) This improvement causes each phase to terminate in a single iteration; Claims 7 and 8 still hold (the other claims are unaffected); and we achieve the time bound stated in Theorem 1.

3. The distributed model. We now consider the following model in the spirit of Papadimitriou and Yannakakis [18], in which distributed agents generate approximate solutions to positive LPs in the standard form presented in section 2.

We associate a *primal agent* with each of the n primal variables y_j and a *dual agent* with each of the m dual variables x_i. Each agent is responsible for setting the value of their associated variable. For any i, j such that $a_{ij} > 0$, we say that dual agent i and primal agent j are *neighbors*. In each distributed round of computation, each agent may broadcast a fixed-size message to all of its neighbors; i.e., in one round each primal agent j may broadcast one message to its set of dual neighbors $I_j = \{i | a_{ij} > 0\}$, and each dual agent i may broadcast one message to its set of primal neighbors $J_i = \{j | a_{ij} > 0\}$.

After a fixed number of rounds, the agents must choose feasible values for their variables to minimize (in the case of the primal) the approximation ratio, $\frac{\text{OPT}}{\sum y_j}$, where OPT is the value of the optimal solution to the LP. We then study the tradeoff between the number of rounds and the quality of the approximation ratio obtained.

The application of flow control in a network with per-flow queuing motivates the following mapping to our model of primal agents and dual agents. Each of n connections transmits data along a fixed path in the network, and a connection corresponds to a primal agent. Each of the paths traverses an ordered subset of the m routers which comprise the network, and these routers correspond to dual agents. At a given time step t, each connection j transmits at a given rate into the network, thereby establishing the value $y_j(t)$ of its primal variable. Once these new flow values stabilize, each router i uses its local load to set a value for the primal variable $x_i(t)$. Based on a simple function (the sum) of the values of dual variables along its path, the source uses this control information to compute a new flow value $y_j(t+1)$. To compute this sum, each connection transmits a fixed-length control message which loops through the routers along its path and back to the source. As mentioned earlier, this simple and natural model of communication between connections and routers corresponds to models previously suggested in both practical and theoretical studies of flow control [3, 1, 5, 16].

Each router i has capacity C_i, which it may share among the connections which utilize it, while each connection accrues benefit B_j for every unit of end-to-end capacity which it receives. Therefore, the connections act as the primal agents and the routers act as the dual agents in the following positive LP:

$$\max \sum_{j=1}^{n} B_j y_j,$$

$$\forall i, \sum_j \tilde{a}_{ij} y_j \leq C_i,$$

$$\forall i, j, \tilde{a}_{ij} = 1 \text{ or } 0.$$

Clearly, this positive LP can be converted to standard form by the local operation $a_{ij} = \frac{\tilde{a}_{ij}}{B_j C_i}$. In a synchronous model, each round takes time equal to the maximum

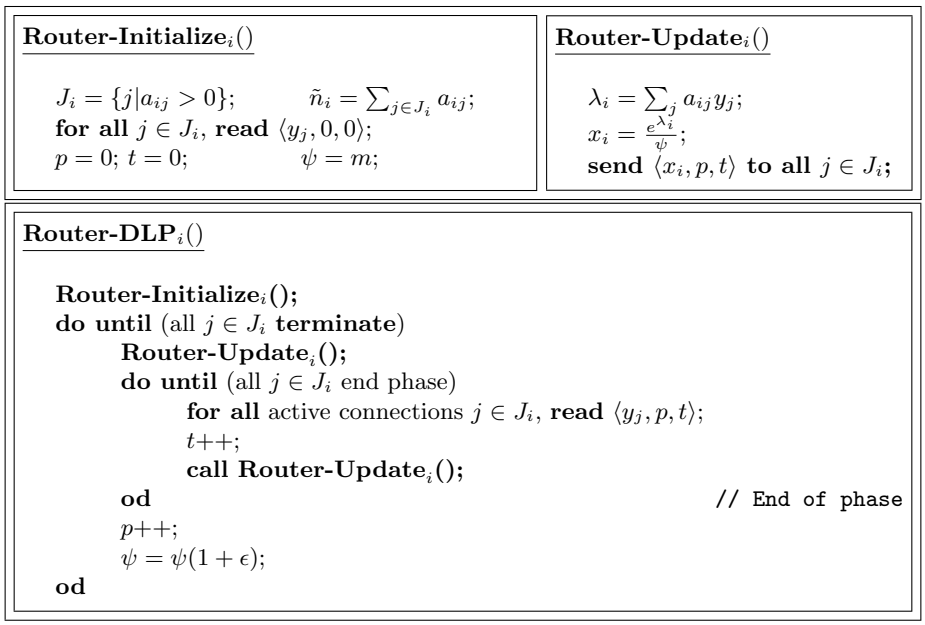


FIG. 3. *The distributed algorithm at router i.*

round-trip time experienced by a connection in the network. However, this synchronization assumption can and will subsequently be relaxed with no changes to the algorithms we propose. A final note is that the message size we use in our implementation can be bounded by a number of bits polynomial in $\log m$, $\log \gamma$, and $1/\epsilon$.

3.1. The distributed approximation algorithm. Several additional complications must be addressed in the definition and description of the distributed algorithm provided in Figures 3 and 4 for routers and connections, respectively. Since global operations cannot be performed efficiently, each connection and router must be able to independently compute the values of all of the parameters described in the serial implementation. In the case of parameters which are fixed, such as the value of m (the number of nodes in the network), and for the parameters which affect the approximation ratio, r and ϵ, we assume that these values are known in advance to all connections and routers. We do not assume that n, the number of connections, is globally known. In the sequential case, knowledge of this parameter was required to initialize the variables y_j so as to satisfy Claims 7 and 8. In the distributed setting, each connection j instead computes a local estimate of n, n_j, which it can compute in two distributed rounds, and which then used in initialization satisfies Claims 7 and 8. Finally, the parameter γ used to convert the program into normalized form may not be globally known, in which case the LP cannot be normalized efficiently. Approximately solving such programs in the distributed setting adds considerable complexity, and we defer providing techniques for doing so until section 3.2.

Connections and routers communicate using the message-passing model described in section 3. As in the serial algorithm, agents track time in terms of phases and iterations. When transmitting the value of a variable using the **send** primitive, agents timestamp the transmission with their current phase number p and iteration t. Likewise, in receiving the value of a variable using the **read** primitive, agents specify

Connection-Initialize$_j()$

$I_j = \{i | a_{ij} > 0\}$;
$n_j = \mathbf{max}_{i \in I_j} \tilde{n}_i$;
$y_j = \frac{\epsilon}{n_j \phi}$;
send$_{i \in I_j}$ $\langle y_j, 0, 0 \rangle$;

Update-$y_j()$

t++;
$y_j = y_j \left(1 + \frac{\epsilon}{\phi}\right)$;
send$_{i \in I_j}$ $\langle y_j, p, t \rangle$;

Compute-$\alpha_j()$

read$_{i \in I_j}$ $\langle x_i, p, t \rangle$;
$\alpha_j = \mathbf{sum}_{i \in I_j} a_{ij} x_i$;

Connection-DLP$_j()$

```
Initialize-Parameters();                    // As defined in Figure 1
Connection-Initialize_j();
p = 0;                                      // Phase counter
do until (ψ > ψ_F)
        t = 0;                              // Iteration counter
        Compute-α_j();
        do until (α_j ≥ 1)
                Update-y_j();
                Compute-α_j();
        od                                  // End of phase
        p++;
od
output y_j and terminate;
```

FIG. 4. *The distributed algorithm at connection j.*

their phase number p and iteration t, and they wait until they receive the appropriate value. For simplicity, we assume that these control messages reliably flow through the path, although, in practice, retransmissions would likely be necessary. Also, strict alternation between primal and dual rounds eliminates the possibility of deadlock.

In our implementation, message-passing primitives enable control to alternate between connections and routers at a local level. This is not to say that control is globally synchronized—in fact, at any instant in time, connections in separate areas of the network might not even be working on the same phase. However, it is the case that any given router is working on only a single phase at an instant in time. Therefore, all the connections through a router which is currently working on phase i are either actively working on phase i themselves or are idle and awaiting permission to proceed to phase $i+1$. Aside from message-passing, the other technical obstacle in converting the centralized algorithm to a distributed algorithm is the condition for ending a phase. In the centralized algorithm, a phase terminates when $\min_k \alpha_k \geq 1$. Since we cannot hope to track the value of this global expression in our distributed model, we instead let each connection j check whether $\alpha_j \geq 1$ locally and independently. When the condition is satisfied, connection j terminates its phase by incrementing its phase number and informing its neighboring routers.

The analysis of feasibility and the bounds on the quality of the approximation are identical to those for the centralized algorithm. This is the case because the value of any primal variable at the time that the corresponding connection completes phase i satisfies the conditions placed on primal variables after phase i in the centralized implementation. A similar statement holds with respect to the values of dual variables at the time their corresponding routers complete phase i. These statements hold for

each primal and dual variable independently, and irrespective of the fact that phase completion times may not occur uniformly across the network. As for the distributed running time, the following corollary to Claim 12 holds.

COROLLARY 13. *The distributed algorithm runs in* $O\left(\frac{(r+1)\ln^2(\frac{\gamma m}{r})\ln\left(\frac{(r+1)\gamma n \ln m}{r\epsilon}\right)}{r^2\epsilon^2}\right)$ *rounds.*

3.2. Distributed techniques to convert to special form. Recall that we can convert a program in standard form to the normalized form by dividing all constraints by a_{max}, thereby setting $\gamma = \frac{a_{max}}{a_{min}}$. If bounds on the values of a_{max} and a_{min} are known in advance, for example, if connections and routers can all bound the min and max values of the edge capacities and benefit coefficients, then γ can be estimated. But these bounds may not be globally known; moreover, the value of γ, which impacts the running time of our algorithm, depends on the values of the entries of the matrix. We now show that solving a problem in standard form can be reduced to solving problems in a *special form* (similar to a form used by Luby and Nisan in [17]), where the value of γ depends only on m and ϵ and does not affect the approximation ratio obtained, nor does it significantly affect the running time of our algorithm. Moreover, this transformation can be done distributively in a constant number of rounds, without global knowledge of a_{max} and a_{min}.

A precondition for transforming an LP Z in standard form to an LP Z' in special form is that we can generate a feasible solution for Z with value c which approximates the value of the optimal solution for Z to within a factor of τ: $c \leq \text{OPT} \leq c\tau$. If this precondition is satisfied, we can perform the following transformation, which bounds the value of the a'_{ij} in Z' by $\frac{\epsilon^2}{m\tau} \leq a'_{ij} \leq 1$ for all i and j, giving $\gamma' = \frac{m\tau}{\epsilon^2}$. Note that the value of γ now depends on the extent to which we can bound the value of the LP, but not on the relative values of the constraints (which could be very small).

Define $\nu = \frac{m}{c\epsilon}$, and perform the following transformation operation on the constraints:

$$a'_{ij} = \begin{cases} \frac{\epsilon}{\nu\tau c} & \text{if } a_{ij} < \frac{\epsilon}{c\tau}, \\ 1 & \text{if } a_{ij} > \nu, \\ \frac{a_{ij}}{\nu} & \text{otherwise.} \end{cases}$$

This transformed LP has the following properties:

1. If $\{y'_j\}$ is a primal feasible solution for Z', then

$$\left\{ y_j = \begin{cases} 0 & \text{if } \exists i \text{ such that } a'_{ij} = 1, \\ \frac{y'_j}{\nu} & \text{otherwise} \end{cases} \right\}$$

   is primal feasible for Z, and $\sum_j y_j \geq \frac{\sum_j y'_j}{\nu} - \epsilon \cdot \text{OPT}$.
2. If $\{y_j\}$ is primal feasible for Z, then $\left\{y'_j = \frac{y_j\nu}{1+\epsilon}\right\}$ is primal feasible for Z', and $\sum_j y'_j = \sum_j \frac{y_j\nu}{1+\epsilon}$.
3. $\frac{\epsilon}{\nu\tau c} \leq a'_{ij} \leq 1$ for all i and j.

Property 3 is clearly true, but the other two properties require the short proofs below.

Proof of property 1. Take a feasible solution $\{y'_j\}$ for Z' and let $\{y_j\}$ be as

specified. Then for any fixed value of i,

$$\sum_j a_{ij} y_j = \sum_{j|a'_{ij}<1} a_{ij} y_j \le \sum_{j|a'_{ij}<1} \nu a'_{ij} y_j \le \sum_{j|a'_{ij}<1} \nu a'_{ij} \frac{y'_j}{\nu} \le 1.$$

The final inequality holds by the feasibility of the solution $\{y'_j\}$, so the solution $\{y_j\}$ is feasible for Z. The value of this solution satisfies

$$\begin{aligned}\sum_j y_j &= \sum_j \frac{y'_j}{\nu} - \sum_{j|\exists i \text{ s.t. } a'_{ij}=1} \frac{y'_j}{\nu} \\ &\ge \sum_j \frac{y'_j}{\nu} - \sum_{j|\exists a'_{ij}=1} \frac{1}{\nu} \\ &\ge \sum_j \frac{y'_j}{\nu} - \frac{m}{\nu} \\ &\ge \sum_j \frac{y'_j}{\nu} - \epsilon \cdot \mathrm{OPT}.\end{aligned}$$

The first inequality holds by the fact that for any feasible $\{y'_j\}$ and for any i and j, $a'_{ij} y'_j \le 1$. The second inequality holds by the bound on the number of routers in the network, and the final inequality holds by the definition of ν. □

Proof of property 2. Take a feasible solution $\{y_j\}$ for Z and let $\{y'_j\}$ be as specified. Then for any fixed value of i,

$$\begin{aligned}\sum_j a'_{ij} y'_j &= \sum_j a'_{ij} \left(\frac{y_j \nu}{1+\epsilon}\right) \\ &= \sum_{j|a_{ij} > \frac{\epsilon}{c\tau}} a'_{ij} \left(\frac{y_j \nu}{1+\epsilon}\right) + \sum_{j|a_{ij} \le \frac{\epsilon}{c\tau}} a'_{ij} \left(\frac{y_j \nu}{1+\epsilon}\right) \\ &\le \sum_{j|a_{ij} > \frac{\epsilon}{c\tau}} \frac{a_{ij} y_j}{1+\epsilon} + \sum_{j|a_{ij} \le \frac{\epsilon}{c\tau}} \frac{\epsilon y_j}{(1+\epsilon)\tau c} \\ &\le \frac{1}{1+\epsilon} + \frac{\epsilon}{(1+\epsilon)\tau c} \sum_j y_j \\ &\le \frac{1}{1+\epsilon} + \frac{\epsilon}{(1+\epsilon)} = 1.\end{aligned}$$

The final line holds from the bound on the optimal solution for Z: $\sum_j y_j \le \mathrm{OPT} \le c\tau$, so the solution y'_j is feasible. □

Now we generate an approximate solution to Z by performing the transformation to special form and then computing a $(1+\epsilon)$ approximation $\{y'_j\}$ for Z' using our algorithm. We transform this solution to $\{y_j\}$ using the transformation described in

property 1 and get a primal feasible solution Y for our LP such that

$$Y = \sum_j y_j \geq \frac{\sum_j y'_j}{\nu} - \epsilon \cdot \mathrm{OPT} \geq \frac{\mathrm{OPT}'}{\nu(1+\epsilon)} - \epsilon \cdot \mathrm{OPT}$$
$$\geq \mathrm{OPT}\left(\frac{1}{(1+\epsilon)^2} - \epsilon\right).$$

The first inequality is from property 1, the second is based on the fact that $\{y'_j\}$ is a $(1+\epsilon)$ approximation to the value of Z' (denoted by OPT'), and the final inequality is from property 2.

Next we need to explain how to choose the parameters c and τ so as to guarantee the precondition $c \leq \mathrm{OPT} \leq c\tau$. Recall that I_j denotes the set of edges incident to connection j, $I_j = \{i | a_{ij} > 0\}$, and J_i denotes the set of connections incident to edge i, $J_i = \{j | a_{ij} > 0\}$. Now define

$$\beta_i = \min_{l \in J_i} \max_{k \in I_l} a_{kl},$$

a quantity which can be locally computed in one round for each router i. Also, let $\beta = \min_i \beta_i$, and for each connection j, define $\widehat{\beta}_j = \min_{i \in I_j} \beta_i$. It is relatively easy to show that $\frac{1}{\beta} \leq \mathrm{OPT} \leq \frac{m}{\beta}$. The first inequality holds from the primal feasibility of the solution in which the connection j used in the evaluation of the minimum β_i is assigned flow $y_j = \frac{1}{\beta}$. The second inequality holds from the dual feasibility of the solution in which each router i is assigned weight $x_i = \frac{1}{\beta_i}$.

Therefore, we can set $c = \frac{1}{\beta}$, and $\tau = m$ in the sequential implementation, giving $\gamma' = \frac{m^2}{\epsilon^2}$ and an $O\big(\frac{nm \ln(m/r\epsilon)}{r\epsilon}\big)$ running time.

3.3. Approximating β in the distributed setting. In the sequential case, knowledge of β is enough to perform the transformation to special form, but connections and routers may not know this value. We now describe a technique in which we distributively subdivide the LP into subprograms based on local estimates of β. The value of each subprogram is bounded, so we can work in special form. Then we recombine solutions in such a way as to only assign nonzero rates to connections with good estimates of β, but we prove that this only reduces the sum of the rates by a small factor.

Set $p = \lceil \frac{1}{\epsilon} \rceil$, and for $q = 0, \ldots, p-1$, define the sets

$$G_t^q = \left\{ j \;\middle|\; \left(\frac{m}{\epsilon}\right)^{p(t-1)+q} \leq \widehat{\beta}_j < \left(\frac{m}{\epsilon}\right)^{pt+q} \right\}$$

for integer t. It is clear that each connection belongs to exactly p of these sets. Independently for each value of q, each router i assigns flow only to connections which are members of $G^q_{T_{iq}}$, where T_{iq} is the minimal value of t for which $G_t^q \bigcap J_i$ is nonempty. In effect, this means that the algorithm is run on the network p successive times. From the connection's point of view, it runs p successive algorithms, using β_j as an approximation for β. In each of these p trials, it can be rejected (i.e., given no flow) by some of the routers. The final flow assigned to connection j is the average of the flows given in the p independent trials. We will prove that this procedure does not decrease the sum of the rates by more than an additional $(1-\epsilon)^2$ factor.

Now define $\mathrm{OPT}(X)$ to be the value of the modified LP when flow can *only* be assigned to connections in the set X. It is not difficult to show that $\mathrm{OPT}(G_t^q)$ is

bounded between $\left(\frac{\epsilon}{m}\right)^{p(t-1)+q}$ and $\left(\frac{\epsilon}{m}\right)^{p(t-1)+q} \cdot \left(\frac{m}{\epsilon}\right)^{1/\epsilon} \cdot m$. Thus, we have that for each set G_t^q, the special form of the modified LP for connections in G_t^q satisfies the precondition with $c = \left(\frac{\epsilon}{m}\right)^{p(t-1)+q}$ and $\tau = \left(\frac{m}{\epsilon}\right)^{1/\epsilon} \cdot m$. Therefore, for each of these LPs, we can use $\gamma = \frac{m\tau}{\epsilon^2} = \left(\frac{m}{\epsilon}\right)^{2+1/\epsilon}$.

We now turn to bound the approximation ratio. Consider a particular $q \in \{0, \ldots, p-1\}$, and let T and Q be the unique integers such that β is in the interval defined by G_T^q and $\beta > \left(\frac{m}{\epsilon}\right)^{pT+Q-1}$. For $q \neq Q$ and for the dual feasible setting $\{x_i = \frac{1}{\beta_i}\}$,

$$\begin{aligned} \mathrm{OPT}\left(\bigcup_{t>T} G_t^q\right) &\leq \sum_{i \mid i \in I_l, l \in G_t^q, t>T} x_i \\ &\leq \frac{m}{\left(\frac{m}{\epsilon}\right)^{pT+Q}} \leq \frac{\epsilon}{\beta} \leq \epsilon \cdot \mathrm{OPT}. \end{aligned}$$

This implies that $\mathrm{OPT}(G_T^q) \geq (1-\epsilon)\,\mathrm{OPT}$ for all $q \neq Q$. The quality of the solution we obtain is therefore bounded below by

$$\frac{1}{p} \sum_{q \neq Q} \mathrm{OPT}(G_T^q) \geq \frac{p-1}{p}(1-\epsilon)\,\mathrm{OPT} \geq (1-\epsilon)^2\,\mathrm{OPT}\,.$$

Putting everything together, we have a distributed algorithm that assumes global knowledge only of m and the approximation parameters r and ϵ. This algorithm finds a primal feasible $(r + (1+\epsilon)^5)$ approximation of the optimal solution, and terminates in $O\left(\frac{(r+1)\ln^2(m/r\epsilon)\ln(\frac{(r+1)mn}{r\epsilon})}{r^2\epsilon^5}\right)$ distributed rounds.

4. Discussion. We studied the problem of having distributed decision-makers with local information generate feasible solutions to positive LPs. Our results explore the tradeoff between the amount of local communication that these agents may perform and the quality of the solution they obtain, measured by the approximation ratio. While we have provided an algorithm which obtains a $(1+\epsilon)$ approximation ratio in a polylogarithmic number of distributed communication rounds, proving nontrivial lower bounds on the running time needed to obtain a $(1+\epsilon)$ approximation remains an open question, as does the challenging problem of providing fast approximation algorithms for general LPs.

Acknowledgments. We would like to thank Christos Papadimitriou for stimulating discussions and useful insights in the early stages of this work. We also thank Dick Karp, Mike Luby, Dorit Hochbaum, and the anonymous SICOMP referees for their helpful comments on earlier versions of results presented in this paper.

REFERENCES

[1] Y. Afek, Y. Mansour, and Z. Ostfeld, *Convergence complexity of optimistic rate based flow control algorithms*, in Proceedings of the 28th ACM Symposium on Theory of Computing, 1996, pp. 89–98.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, *On-line load balancing with applications to machine scheduling and virtual circuit routing*, in Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, 1992, pp. 164–173.

[3] B. Awerbuch and Y. Azar, *Local optimization of global objectives: Competitive distributed deadlock resolution and resource allocation*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 1994, pp. 240–249.

[4] B. Awerbuch, Y. Azar, and S. Plotkin, *Throughput competitive on-line routing*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, 1993, pp. 32–40.

[5] B. Awerbuch and Y. Shavitt, *Converging to approximated max-min flow fairness in logarithmic time*, in Proceedings of the 17th IEEE INFOCOM Conference, 1998, pp. 1350–1357.

[6] Y. Bartal, J. Byers, and D. Raz, *Achieving global objectives using local information with applications to flow control*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 303–312.

[7] D. Bertsekas and R. Gallagher, *Data Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[8] D. Bienstock, *Potential function methods for approximately solving linear programming problems: Theory and practice*, in CORE Lecture Series; available from CORE, University Catholique de Louvain, Belgium.

[9] F. Bonomi and K. Fendick, *The rate-based flow control framework for the available bit rate ATM service*, IEEE Network Magazine, 9 (2) (1995), pp. 24–39.

[10] J. Byers, *Maximizing Throughput of Reliable Bulk Network Transmissions*, Ph.D. Thesis, University of California at Berkeley, Berkeley, CA, 1997.

[11] A. Charny, *An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback*, Technical Report MIT/LCS/TR-601, MIT Laboratory for Computer Science, Cambridge, MA, 1994.

[12] N. Garg and J. Koenemann, *Faster and simpler algorithms for multicommodity flow and other fractional packing problems*, in Proceedings of the 39th Annual Symposium on Foundations of Computer Science, 1998, pp. 300–309.

[13] J. Jaffe, *Bottleneck flow control*, IEEE Trans. Comm., 29 (1981), pp. 954–962.

[14] H. Karloff, *Linear Programming*, Birkhäuser, Boston, MA, 1996.

[15] N. Linial, *Distributive graph algorithms—global solutions from local data*, in Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 331–335.

[16] S. Low and D. Lapsley, *Optimization flow control* I: *Basic algorithm and convergence*, IEEE/ACM Trans. Networking, 7 (1999), pp. 861–874.

[17] M. Luby and N. Nisan, *A parallel approximation algorithm for positive linear programming*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993, pp. 448–457.

[18] C. Papadimitriou and M. Yannakakis, *Linear programming without the matrix*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993, pp. 121–129.

[19] S. Plotkin, D. Shmoys, and E. Tardos, *Fast approximation algorithms for fractional packing and covering problems*, in Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 495–504. Full version appears as Technical Report ORIE-999, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1995.

SIAM J. COMPUT.
Vol. 33, No. 6, pp. 1280–1308

PREEMPTIVE WEIGHTED COMPLETION TIME SCHEDULING OF PARALLEL JOBS*

UWE SCHWIEGELSHOHN†

Abstract. We present a new algorithm for the preemptive offline scheduling of independent jobs on a system consisting of m identical machines. The jobs can be parallel; that is, they may need the concurrent availability of several machines for their execution. To this end, we introduce a machine model which is based on existing multiprocessors and accounts for the penalty of preemption. After examining the relation between makespan and total weighted completion time costs for the scheduling of parallel jobs, we show that our new algorithm achieves an approximation factor of 2.37 for total weighted completion time scheduling if no preemption penalty is considered. This compares favorably to the thus far best approximation factor of 8.53 for the nonpreemptive case. To fine-tune the algorithm with respect to different preemption penalties, we use a fairly simple numerical optimization problem. Further, we present an algorithm to transform the preemptive schedule into a nonpreemptive one. This leads to an improved approximation factor of 7.11 for the nonpreemptive weighted completion time scheduling.

Key words. scheduling, approximation algorithms

AMS subject classifications. 68M20, 68Q25, 90B35

DOI. 10.1137/S009753979731501X

1. Introduction.

We address preemptive and nonpreemptive scheduling of parallel jobs without release dates. There is a job system τ consisting of independent parallel jobs to be scheduled on m identical machines. Each job j is characterized by its fixed degree of parallelism $m_j \leq m$, its processing time $p_j \geq 1$, and a weight $w_j \geq 0$. Exactly m_j machines must be allocated to a job j at any time the job is executing and each machine can execute at most one job at a time. Note that the actual processing of job j will require the same time on any subset of m_j machines. In the preemptive version, it is possible to temporarily suspend the execution of a job j on all machines assigned to it and to resume its processing on the same subset of machines at a later time, that is, no migration of the job after a preemption is permitted. This preemption process can be repeated. Both the suspension and the resumption of the job will result in a preemption penalty of $\frac{p}{2}$. Our preemptive model and its relation to real implementations are explained in section 2 in more detail.

The completion time of job j in a schedule S is denoted as $C_j(S)$. In this paper, it is the goal of the scheduling algorithm to minimize the sum of the weighted completion times $C(S) = \sum_{j\in\tau} w_j C_j(S)$ of S, also called total weighted completion time of S. These problems are denoted as $Pm|\text{size}_j|\sum w_j C_j = Pm|m_j|\sum w_j C_j$ and $Pm|m_j, \text{prmp, no mig}|\sum w_j C_j$, respectively. In addition, we consider the makespan $C_{\max}(S) = \max_{j\in\tau} C_j(S)$ of S, that is, the problems $Pm|m_j|C_{\max}$ and $Pm|m_j, \text{prmp, no mig}|C_{\max}$. Minimizing the total weighted completion time is known to be NP-hard in the strong sense, even if all jobs are sequential; that is, $m_j = 1$ is

*Received by the editors January 15, 1997; accepted for publication (in revised form) December 22, 2003; published electronically August 6, 2004. This research was supported by the NRW Metacomputing grant. A preliminary version of this paper appeared in *Proceedings of the European Symposium of Algorithms ESA* '96, Barcelona, Spain, Lecture Notes in Comput. Sci. 1136, Springer-Verlag, Berlin, 1996, pp. 39–51.

http://www.siam.org/journals/sicomp/33-6/31501.html

†Computer Engineering Institute, University Dortmund, 44221 Dortmund, Germany (Uwe.Schwiegelshohn@udo.edu).

valid for all $j \in \tau$ [2]. This result also holds in the preemptive case [9]. Hence, we are interested in polynomial-time algorithms that provide approximate solutions. Such an algorithm is called a ρ-approximation algorithm if its result is never worse than ρ times the optimal value. Then we also say that ρ is the approximation factor. Further, we denote the optimal total weighted completion time and the optimal makespan for a job system τ on a given machine system $\mathcal{M}$ as $C^*(\tau, \mathcal{M})$ and $C^*_{\max}(\tau, \mathcal{M})$, respectively. If the context is clear, we will omit τ and/or $\mathcal{M}$.

Approximation algorithms have already been presented for several nonpreemptive variants of this problem. For sequential jobs ($m_j = 1$), Kawaguchi and Kyan [8] described a list scheduling algorithm called LRF (*largest ratio first*) with an approximation factor of $\frac{1+\sqrt{2}}{2}$. This problem is denoted as $Pm||\sum w_j C_j$. The extension of this problem to include parallel jobs ($Pm|m_j|\sum w_j C_j$) has only recently been addressed when Schwiegelshohn et al. [11] showed that the SMART (*scheduling to minimize average response time*) algorithm generates shelf-based nonpreemptive schedules with an approximation factor of 8.53. Turek et al. [14] proved that a generalization of Kawaguchi and Kyan's LRF method produces an approximation factor of 2 for parallel jobs with unique weights if the resource requirement of each job is at most 50% of the maximum number of machines. However, when allowing arbitrary jobs, this method may result in schedules which significantly deviate from the optimum.

Among other problems, Chakrabarti et al. [3] addressed the nonpreemptive scheduling of so-called malleable jobs with different release dates and proved an expected performance within 8.67 of optimal for a randomized online algorithm. We call a job malleable if it can be executed on any number of machines up to m_j with a corresponding increase or decrease of the processing time. Deng et al. [5] discussed preemptive response time scheduling for malleable jobs with unique weights and described a 2-approximation algorithm. They use an online model where the total resource consumption of a job (the sum of the execution times of this job on all machines) is invariant but becomes known only at the completion of the job. Note that Deng's model is significantly different from ours as migration is allowed and the number of machines allocated to a job may change after the preemption of the job. Based on the work from Afrati et al. [1], Fishkin, Jansen, and Porkolab [7] recently showed a PTAS (polynomial time approximation scheme) for the problem $Pm|m_j, r_j|\sum w_j C_j$ if the number of machines is constant; that is, a $1+\epsilon$ approximation is achieved with a running time of $f(\frac{1}{\epsilon}, m)O(n \log n)$. Note that the running time of this algorithm is exponential in m while m is used in our algorithm only to compare values. Finally, Phillips, Stein, and Wein [10] have demonstrated how to transform preemptive schedules of sequential jobs ($m_j = 1$) into nonpreemptive schedules with only a factor 3 increase in the average weighted completion time.

In this paper, we present a 2.37-approximation algorithm for our preemptive problem if there is no preemption penalty. This result is further used to derive a nonpreemptive algorithm with an approximation factor of 7.11. In both cases, we also give a performance guarantee for the makespan problem. This concept of bicriteria scheduling has been introduced by Stein and Wein [13] who showed that there are always schedules which approximate both the makespan and the total weighted completion time.

After presenting our model in section 2 and comparing it to some real world constraints, we discuss some aspects of bicriteria scheduling of parallel and independent jobs. Previous results with relevance to our algorithm are addressed in section 4. Next, we introduce the new preemptive algorithm in section 5 and analyze it after-

ward. We show that the total weighted completion time approximation factor of our algorithm can be minimized by solving a numerical optimization problem. Then we present a method, similar to the one published by Phillips, Stein, and Wein [10], to transform our preemptive schedule into a nonpreemptive one. Finally, our preemptive and nonpreemptive results are compared to the worst-case bounds of nonpreemptive SMART schedules.

2. The model. For a given job system τ and a machine system $\mathcal{M}$ consisting of m independent machines and a preemption penalty p, we now formally introduce a preemptive schedule with no migration being permitted.

DEFINITION 2.1 (preemptive schedule without migration).

In a preemptive schedule $S(\tau, \mathcal{M})$ without migration, each job $j \in \tau$ is assigned

1. *a nonnegative integer d_j,*
2. *a $(2d_j + 2)$-tuple of time instants $t_j(0) \le t_j(1) \le \cdots \le t_j(2d_j + 1)$, and*
3. *a set $M_j \subseteq \{1, \ldots, m\}$ with $|M_j| = m_j$.*

Again, we usually omit τ and $\mathcal{M}$. In Definition 2.1, d_j denotes the number of preemptions for job j in schedule S. The execution of job j in S is started at time $t_j(0)$, suspended at times $t_j(2\delta - 1)$, resumed at times $t_j(2\delta)$, and completed at time $t_j(2d_j+1) = C_j(S)$ with $0 < \delta \le d_j$. Finally, M_j is the set of machines on which job j is executed. Note that for a system of identical machines and either a nonpreemptive schedule or a preemptive schedule with migration, it is sufficient to consider only the number m_j of required machines, while the specification of the actual machine set M_j is necessary in a preemptive schedule without migration. In the latter case, we say that a job j is resident on M_j from $t_j(0)$ to $t_j(2d_j + 1)$.

A preemptive schedule $S(\tau, \mathcal{M})$ without migration is called valid if the following conditions are fulfilled for all $j \in \tau$:

1. $\sum_{\delta=0}^{d_j} t_j(2\delta + 1) - t_j(2\delta) = p_j + pd_j$,
2. $t_j(\mu) - t_j(\mu - 1) \ge \begin{cases} 0 & \text{for } \mu = 2\delta \text{ with } 0 < \delta \le d_j, \\ \frac{p}{2} & \text{for } (\mu = 1 \text{ or } \mu = 2d_j + 1) \text{ and } d_j \ne 0, \\ p & \text{for } \mu = 2\delta + 1 \text{ with } 0 < \delta < d_j, \end{cases}$
3. $M_j \cap M_{j'} = \emptyset$ if $j \ne j'$ and $t_j(2\delta) \le t_{j'}(2\delta') < t_j(2\delta + 1)$ for any $0 \le \delta \le d_j$ and $0 \le \delta' \le d_{j'}$

Job j executes between time $t_j(2\delta)$ and time $t_j(2\delta+1)$ while it is inactive between time $t_j(2\delta - 1)$ and time $t_j(2\delta)$. As each suspension and each resumption of a job execution produce a preemption penalty of $\frac{p}{2}$, there is a minimal time period between resumption and the next suspension of a job. This property is described in condition 2 above. Note that no preemption penalty is encountered if a job starts or terminates. The various cases of condition 2 can be seen in Figure 1. In the figures shown in this paper, all machines are arranged along a single (horizontal) line. In a nonpreemptive schedule, the execution of a job can therefore be described as several rectangles of the same height which are aligned along a horizontal line. However, for reasons of simplicity, we simply assume that the machines assigned to a job are arranged in a contiguous fashion. Hence, we represent each nonpreemptively executed job as a single rectangle. Then in a preemptive schedule without migration, the execution of a job is described as several rectangles which are aligned along a vertical line. In order to better demonstrate the preemption penalty, we assume that during the preemption penalty all involved machines remain idle; see Figure 1. Therefore, the combined area of all rectangles associated with the actual processing of a job j remains invariant and has the value $m_j p_j$. Condition 1 gives the total time a job j is active on its assigned machines including all preemption penalties. Finally, condition 3 ensures

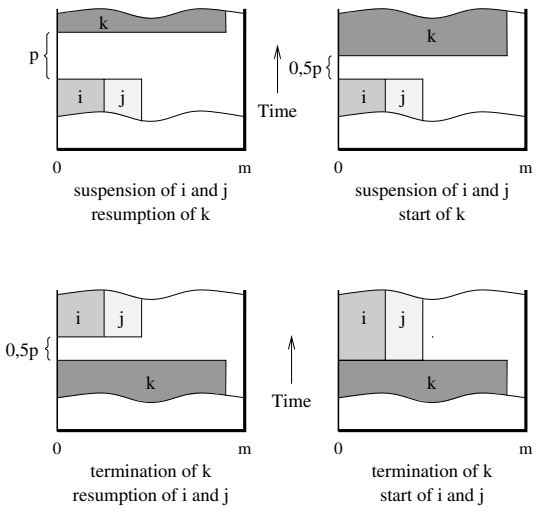


FIG. 1. *Different preemption penalty cases.*

that no machine can be assigned to more than one job at any time.

Our machine model is based on multiprocessor systems like the IBM RS/6000 SP. A typical installation consists of m nodes which use an interconnection network for communication. Each node has its own processor, main memory, and a local hard disk for swapping. Usually most of the nodes are identical. In the IBM machine, the interconnection network does not prefer specific subsets of nodes, and the nodes are assigned to jobs in an exclusive fashion. Therefore, the execution time of a job does not depend on the assigned node partition.

Some multiprocessors perform preemption by simultaneously switching the context of all nodes executing the same job. This kind of preemption is called *gang scheduling* [6]. Our model also uses gang scheduling as at any time, a job can execute only on all or on none of the machines of its machine set. In practice, the context switch is typically executed with the help of the local processor memory and/or the local hard disk. Typically during the context switch, the interconnection network is needed only to synchronize the nodes of a partition to act concurrently. However, the context switch may leave some messages in the interconnection network without target process. These messages must be temporarily saved until the target process resumes execution. Further, the context switch requires the saving and loading of a job status and may cause additional cache misses and page faults. In our model, all those delays are combined into a constant overhead, the so-called preemption penalty $\frac{p}{2}$. This preemption penalty is encountered whenever the execution of a job is interrupted or resumed. Therefore, a complete context switch produces the preemption penalty p.

3. Bicriteria scheduling. Most previous scheduling work has addressed a single scheduling goal. Algorithms that approximate more than one criterion have received little attention. Recently, Stein and Wein [13] showed that for any scheduling problem, there are always schedules with a total weighted completion time and a makespan that are both within 2 of the optimal value.

Although we focus mainly on the total weighted completion time in this paper, we also address the makespan of our schedules. While Stein and Wein were primarily

interested in the existence of schedules with bicriteria properties, we want to show that our polynomial time algorithms approximate both the total weighted completion time and the makespan, similar to Chakrabarti et al. [3].

In this section, we give a few results to show the relation between total weighted completion time and makespan scheduling for parallel jobs on identical machines. First, we present a simple example to demonstrate that there are job systems where any optimal makespan schedule $S_{\max}$ results in $C(S_{\max}) = \Theta(m)C^*$.

Example 3.1. Assume a system τ of $m+1$ jobs such that there are m identical jobs j with $m_j = 1$, $p_j = 1$, and $w_j = 1$ and a single job j' with $m_{j'} = m-1$, $p_{j'} = m$, and $w_{j'} = 1$. It is easy to see that in any optimal makespan schedule $S_{\max}$, job j' must start at time 0 on any subset of $m-1$ machines while all other jobs are executed one after another on the single remaining machine. However, in the optimal total completion time schedule S, all m small jobs start concurrently at time 0 while job j' is executed after all other jobs are completed. This yields $C_{\max}(S_{\max}) = m = C^*_{\max}$ and $C(S_{\max}) = \frac{m(m+3)}{2} = \Theta(m)(2m+1) = \Theta(m)C(S)$. Although $S_{\max}$ and S are both nonpreemptive schedules, the result is also valid in the preemptive case.

Next, we address the opposite case; that is, we determine the makespan for optimal total weighted completion time schedules. First, we consider nonpreemptive schedules.

LEMMA 3.1. *There are job systems such that any nonpreemptive optimal total weighted completion time schedule S results in $C_{\max}(S) = \Theta(m)C^*_{\max}$.*

Proof. Assume a system of $2m$ jobs such that $p_{2i-1} = m+i-1$, $m_{2i-1} = 1$, $p_{2i} = 1$, and $m_{2i} = m$ hold for $1 \le i \le m$. The weights of the jobs are defined recursively. We start with $w_{2m} = 1$. For all other jobs, there is $w_{2i-1} = w_{2i}(m+i)$ and $w_{2i} = (m+i)\sum_{k=2i+1}^{2m} w_k$.

It is easy to see that we have $C^*_{\max} = 3m-1$ as job $2m-1$ cannot be executed concurrently with any job $2i$. Therefore, an optimal makespan schedule is obtained by starting job $2i$ at time i while all jobs $2i-1$ concurrently start their execution at time m.

On the other hand, we consider a list schedule S where all jobs start in the order $1, 2, \ldots, 2m$. Then no two jobs are executed concurrently as each sequential job ($m_j = 1$) is sandwiched between two parallel jobs requiring all machines. This results in $C_{\max}(S) = 1.5(m^2+m) = \Theta(m)C^*_{\max}$. It remains to be shown that S is the only nonpreemptive schedule with optimal total weighted completion time.

Note that we need to consider only schedules with nonnegative integer starting times for all jobs as the processing time of each job is an integer. Otherwise it is possible to reduce the total weighted completion time by starting the first job j with a noninteger starting time r at time $\lfloor r \rfloor$, as this does not affect the execution of any other job.

Assume that the starting times of jobs $1, \ldots, 2k$ are identical in schedule S and in any optimal total weighted completion time schedule. Then no job $j > 2k$ can start before $C_{2k}(S) = t_{2k}(1)$ due to its processing time for odd j or its degree of parallelism for even j. Therefore, we simply disregard all jobs $1, \ldots, 2k$ and assume $t_{2k}(1) = 0$ for the rest of the proof. Let us further introduce job system $\check{\tau} = \{2k+3, \ldots, 2m\}$. There are three possible cases:

1. *Schedule S_1.* Jobs $2k+1$ and $2k+2$ start at times 0 and $p_{2k+1} = m+k$, respectively. All jobs from $\check{\tau}$ are executed in the time frame $[m+k+1, \infty)$ in an optimal fashion.

2. *Schedule S_2.* Job $2k+1$ does not start before time 1. All jobs from $\check{\tau}$ and

job $2k+2$ are executed in the time frame $[0,\infty)$ in an optimal fashion.

3. *Schedule* S_3. Job $2k+2$ does not start before time $p_{2k+1}+1$. All jobs from $\check{\tau}$ and job $2k+1$ are executed in the time frame $[0,\infty)$ in an optimal fashion.

This results in the following total weighted completion times:

$$\begin{aligned}
C(S_1) &= C^*(\check{\tau}) + \left(\sum_{j\in\check{\tau}} w_j\right)(m+k+1) + w_{2k+2}(m+k+1)(m+k)\\
&\quad + \ w_{2k+2}(m+k+1)\\
&= C^*(\check{\tau}) + \left((m+k+1)^2 + \frac{m+k+1}{m+k+2}\right) w_{2k+2},\\
C(S_2) &\geq C^*(\check{\tau}) + w_{2k+2}(m+k+1)(m+k+1) + w_{2k+2}\\
&= C^*(\check{\tau}) + ((m+k+1)^2+1)w_{2k+2},\\
C(S_3) &\geq C^*(\check{\tau}) + w_{2k+2}(m+k+1)(m+k) + w_{2k+2}(m+k+2)\\
&= C^*(\check{\tau}) + ((m+k+1)^2+1)w_{2k+2}.
\end{aligned}$$

Therefore, jobs $2k+1$ and $2k+2$ must start at times $t_{2k}(1)$ and $t_{2k}(1)+p_{2k+1}$, respectively, to obtain an optimal total weighted completion time schedule. It follows by induction that S is the only schedule with optimal total weighted completion time. □

However, if preemption with migration is allowed, the relation between total weighted completion time scheduling and makespan scheduling is different as shown by the next lemma. Contrary to the model of Definition 2.1, the model of Lemma 3.2 allows the execution of job j to resume after a preemption on any subset of m_j idle machines. This problem is denoted as $P|m_j, \text{prmp}|\sum w_jC_j$.

LEMMA 3.2. *Assume m identical machines and a system τ of independent jobs. If preemption with job migration and without preemption penalty is allowed, then any optimal total weighted completion time schedule S also guarantees $C_{\max}(S) \leq (2-\frac{1}{m})C^*_{\max}$.*

Proof. We say that a time instance t in schedule S is *open* if at most $\frac{m}{2}$ machines are used at t. If schedule S does not contain any open time instances in the interval $[0, C_{\max}(S))$, then we have

$$C^*_{\max} \geq \frac{1}{m}\sum_{j\in\tau} m_jp_j \geq \frac{m+1}{2m}C_{\max}(S) \geq \frac{m}{2m-1}C_{\max}(S).$$

Otherwise, assume that t is the last open time instance in schedule S with m_t machines being busy at t. Further, let job j execute at t. Then S has the following properties:

(i) If j does not execute in S at a time instance t' of the interval $[0, t_j(2d_j+1))$, then at least $m-m_j+1 \geq m-m_t+1$ machines are busy at t'.

(ii) If j executes at a time instance t' in S, then at least $m_t \geq m_j$ machines must be used at t' as all jobs executing at t must also execute at all previous open time instances.

(iii) At least $m-m_t+1$ machines must be used at any time instance t' in the interval $[C_j(S), C_{\max}(S))$ as t is the last open time instance and there must be at least one job that executes at t' while not executing at t if we have $C_j(S) < C_{\max}(S)$.

Therefore, a total machine-time product of at least $m_tp_j+(m-m_t+1)(C_{\max}(S)-$

$p_j) \leq mC^*_{\max}$ is required for τ. For $p_j \geq \frac{C_{\max}(S)}{2-\frac{1}{m-m_t+1}}$, this results in

$$\frac{C_{\max}(S)}{2-\frac{1}{m}} \leq \frac{C_{\max}(S)}{2-\frac{1}{m-m_t+1}} \leq p_j \leq C^*_{\max}.$$

Otherwise as $2m_t < m+1$ holds, we have

$$\begin{aligned} C^*_{\max} &\geq \frac{1}{m}((m-m_t+1)C_{\max}(S) - (m-2m_t+1)p_j) \\ &\geq \left(m-m_t+1-\frac{m-2m_t+1}{2-\frac{1}{m-m_t+1}}\right)\frac{C_{\max}(S)}{m} \\ &= \frac{1}{2-\frac{1}{m-m_t+1}}C_{\max}(S) \geq C_{\max}(S)\frac{1}{2-\frac{1}{m}}. \qquad \square \end{aligned}$$

This lemma shows that any optimal total weighted completion time schedule deviates by at most a factor $2-\frac{1}{m}$ from the optimal makespan schedule for a specific scheduling problem $(P|m_j, \text{prmp}|\sum w_jC_j)$, while Stein and Wein [13] proved a more general result with a deviation factor for the total weighted completion time. Also note that it is still an open problem to bound the makespan approximation factor for an optimal total weighted completion time schedule using our model of preemption without migration.

4. Previous results and simple extensions. We start this section by extending our comparison of preemptive and nonpreemptive schedules with respect to the total weighted completion time criterion. Then we give the total weighted completion time and the makespan approximation factors for two types of algorithms: a list scheduling algorithm based on Smith's ratio [12] and the SMART algorithm [15].

For the case that all jobs of a job system are sequential ($m_j = 1$ for all $j \in \tau$), McNaughton [9] proved that the total weighted completion time of the optimal nonpreemptive schedule cannot be reduced by introducing preemption. However, our results in the previous section already indicate that there are differences between preemptive and nonpreemptive schedules if jobs are parallel. The following simple example shows that the total weighted completion time of an optimal preemptive schedule is smaller than the corresponding value of an optimal nonpreemptive schedule for some job systems. It can also be seen that this result holds, even if all jobs have unit weight.

Example 4.1. Assume a system of three jobs with the properties $w_1 = w_2 = w_3 = 1$, $p_1 = 2$, $p_2 = 1$, $p_3 = 4$, $m_1 = m$, $m_2 = m-1$, and $m_3 = 1$. In an optimal nonpreemptive total weighted completion time schedule S_n, job 2 starts concurrently with job 3. This block and job 1 are then scheduled in any order resulting in $C(S_n) = 11$; see the left schedule of Figure 2.

In the optimal preemptive total weighted completion time schedule S_p, however, first job 2 and job 3 are started together; see the right schedule of Figure 2. Then at time $t = 1$, job 3 is interrupted and job 1 is executed to completion. Finally, the execution of job 3 is resumed at time $t = 3$. Assuming no preemption penalty ($p = 0$), we have $C(S_p) = 10$.

Next, we consider the LRF algorithm, a simple nonpreemptive list scheduling algorithm where the order of the jobs is determined by the nonincreasing ratio $s_j = \frac{w_j}{p_j m_j}$. Since the ratio $\frac{w_j}{p_j}$ is known as Smith's ratio [12], we call s_j the modified Smith ratio. The LRF algorithm always removes the first job in a list and schedules it as

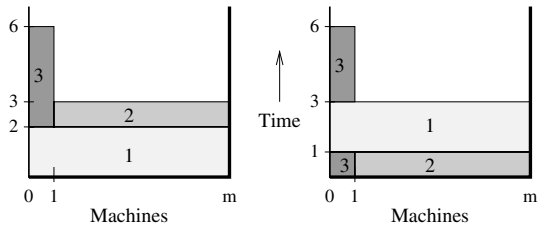


FIG. 2. *Optimal nonpreemptive schedule (left) and optimal preemptive schedule (right).*

early as possible without affecting the completion time of any previously scheduled job.

Kawaguchi and Kyan [8] have shown that the LRF algorithm guarantees $C(S) \leq \frac{\sqrt{2}+1}{2} C^*$ if all jobs are sequential ($m_j = 1$ for all $j \in \tau$).

Later, Turek et al. [14] proved that the approximation factor of the LRF algorithm is 2 if $m_j \leq \frac{m}{2}$ and $w_j = 1$ hold for each job $j \in \tau$. This proof can be easily extended to the weighted case. Before this is done, we repeat Lemma 3.2 of [11], which gives several lower bounds for nonpreemptive weighted completion time schedules.

LEMMA 4.1. *For any job system τ and a machine system that does not allow preemption, the optimal total weighted completion time solution C^* satisfies $A_\tau \leq C^*$, $\sum_{j\in\tau} w_j p_j \leq C^*$, and $A_\tau + \frac{1}{2}\sum_{j\in\tau} w_j p_j - \frac{1}{2m}\sum_{j\in\tau} w_j p_j m_j \leq C^*$.*

A_τ is called the squashed area bound and was originally introduced for the unweighted case in [15]. For this bound, each job j is transformed into a fully parallel job j' with $m_{j'} = m$, $p_{j'} = \frac{p_j m_j}{m}$, and $w_{j'} = w_j$. Therefore, the resulting scheduling problem is equivalent to a simple single machine problem, and an optimal schedule for these parallel jobs is generated by Smith's rule. For a proof of Lemma 4.1, see [11].

Next, we extend Turek's Lemma 4.1 in [14] to consider arbitrary weights as well.

LEMMA 4.2. *For any job system τ with $m_j \leq \frac{m}{2}$ for all $j \in \tau$ and a machine system that does not allow preemption, $C(S) \leq 2C^*$ holds for all LRF schedules S.*

Proof. We prove this lemma by induction. Let us assume that the jobs of τ are enumerated using the modified Smith ratio and that the following condition holds for job system $\tau' = \{1, \ldots, j-1\}$:

$$C(\tau') \leq 2A_{\tau'} + \sum_{i\in\tau'} w_i p_i - \frac{1}{m}\sum_{i\in\tau'} w_i p_i m_i \leq 2C^*(\tau').$$

Using $\tau'' = \tau' \cup \{j\}$, we need to show that

$$\begin{aligned} C(\tau'') &= C(\tau') + w_j C_j(S) \\ &\leq 2A_{\tau''} + \sum_{i\in\tau''} w_i p_i - \frac{1}{m}\sum_{i\in\tau''} w_i p_i m_i \\ &= 2A_{\tau'} + 2\frac{w_j}{m}\sum_{i\in\tau'} p_i m_i + 2\frac{w_j p_j m_j}{m} + \sum_{i\in\tau'} w_i p_i + w_j p_j - \frac{1}{m}\sum_{i\in\tau'} w_i p_i m_i \\ &\quad - \frac{w_j p_j m_j}{m}. \end{aligned}$$

Due to the induction assumption, this is true if the following inequality holds:

$$w_j(C_j(S) - p_j) \leq 2\frac{w_j}{m}\sum_{i\in\tau'} p_i m_i + \frac{w_j p_j m_j}{m}.$$

To see the validity of the inequality, note that $C_j(S) - p_j$ is the starting time of job j in any nonpreemptive schedule S. Further, at any time instance before $C_j(S) - p_j$, more than $\frac{m}{2}$ machines are used by jobs in τ' as S is a list schedule and each job needs at most $\frac{m}{2}$ machines. ∎

Next, let us consider the case where all jobs of a job system require more than 50% of the machines ($m_j > \frac{m}{2}$). Here it is easy to obtain an optimal total weighted completion time schedule as no two jobs can be executed concurrently. However, due to different machine requirements of the various jobs, scheduling the jobs in Smith order is not necessarily optimal.

COROLLARY 4.3. *Assume a job system τ with $m_j > \frac{m}{2}$ for all $j \in \tau$. Then any LRF schedule S satisfies $C(S) \leq 2C^*$.*

Proof. We construct a new job system τ' such that for each job $j \in \tau$, there is a job $j' \in \tau'$ with $w_{j'} = w_j$, $m_{j'} = m$, and $p_{j'} = \frac{p_j m_j}{m} > \frac{p_j}{2}$. Arranging the jobs of τ' in Smith order produces an optimal schedule $S' = A_{\tau'}$. Note that jobs j and j' have the same relative position in schedules S and S', respectively. This results in $C_j(S) < 2C_{j'}(S')$ and leads to

$$C(S) < 2C(S') = 2A_{\tau'} = 2A_\tau \leq 2C^*. \qquad \blacksquare$$

But if both cases in Lemma 4.2 and Corollary 4.3 are combined, meaning that jobs may be arbitrary, the approximation factor for the LRF algorithm may be as bad as m [14].

Further, many nonpreemptive schedules with small approximation factors for $C^*_{\max}$ cannot guarantee a constant approximation factor for C^*; see [15]. Here, we want to address the opposite case and discuss the approximation factors for $C^*_{\max}$ of some nonpreemptive schedules with good total weighted completion time performance. For the sake of completeness, we briefly recall the γ-SMART$_{\text{NFIW}}$ as used in [11]. The SMART algorithm generates a shelf schedule; that is, the jobs are assigned to shelves, and all jobs belonging to the same shelf start at the same time. The height of a shelf is the processing time of the longest running job assigned to it, and a shelf is placed on top of its predecessor with the first shelf starting at time 0. For γ-SMART, the job system τ is partitioned such that all jobs j with $\gamma^\mu \leq p_j < \gamma^{\mu+1}$ belong to the same height component $\mu \geq 0$. The jobs in such a height component are then ordered by increasing ratio $\frac{m_j}{w_j}$ and assigned to shelves in a next fit fashion (next fit increasing width (NFIW) to weight). Finally, the shelves are scheduled using Smith's rule, where the Smith ratio of a shelf is determined by its height and the sum of the weights of all jobs assigned to it.

COROLLARY 4.4. *For any job system τ, a γ-SMART$_{NFIW}$ schedule S satisfies $C_{\max}(S) < (1 + 2\gamma + \frac{\gamma}{\gamma-1})C^*_{\max}$.*

Proof. First, we restrict ourselves to a job system $\tau_1 \subseteq \tau$ that contains all jobs scheduled on the first shelf of each height component. We use the notation $C_{\max}(S, \tau_1)$ to denote the makespan of schedule S if we restrict ourselves to the jobs of τ_1 and remove all time intervals of S where all machines are idle. Assuming $\gamma^k \leq p_{\max} = \max_{j\in\tau} p_j < \gamma^{k+1}$, that is, at most $k+1$ height components, we obtain for the makespan $C_{\max}(S, \tau_1) \leq p_{\max} + \gamma\frac{\gamma^k - 1}{\gamma - 1}$. Therefore, we have

$$C_{\max}(S, \tau_1) < \left(1 + \frac{\gamma}{\gamma - 1}\right) p_{\max} \leq \left(1 + \frac{\gamma}{\gamma - 1}\right) C^*_{\max}(\tau).$$

Next, we consider all shelves belonging to the same height component μ. Assume that those shelves are enumerated in the order in which they are filled with jobs. As

the jobs of this height component are assigned to the shelves in a next fit fashion, the width of the first job in shelf $2i$ plus the width of all jobs in shelf $2i-1$ must exceed m. With n_μ and τ_μ denoting the number of shelves and the jobs in height component μ, respectively, we obtain

$$\sum_{j\in\tau_\mu} m_j > \left\lfloor \frac{n_\mu}{2} \right\rfloor m \geq \frac{n_\mu - 1}{2} m$$

and

$$\frac{1}{m} \sum_{j\in\tau_\mu} p_j m_j > \frac{n_\mu - 1}{2} \gamma^\mu.$$

Note that $n_\mu \gamma^{\mu+1}$ is an upper bound for the combined height of all shelves of height component μ in schedule S. Further, one shelf of each height component is already considered in the schedule for the jobs of τ_1. Finally, we have $C^*_{\max}(\tau) \geq \frac{1}{m}\sum_{j\in\tau} p_j m_j$. The combination of all those results leads to

$$\begin{aligned} C_{\max}(S) &< C_{\max}(S,\tau_1) + \sum_{\mu=0}^{k} (n_\mu - 1)\gamma^{\mu+1} < \left(1 + \frac{\gamma}{\gamma - 1}\right) C^*_{\max}(\tau) + 2\gamma \frac{1}{m} \sum_{j\in\tau} p_j m_j \\ &< \left(1 + \frac{\gamma}{\gamma - 1} + 2\gamma\right) C^*_{\max}(\tau). \qquad \square \end{aligned}$$

In the next corollary, we again address job systems where no job needs more than 50% of the machines.

COROLLARY 4.5. *Assume a job system τ with $m_j \leq \frac{m}{2}$ for all $j \in \tau$. Then an LRF schedule S satisfies $C_{\max}(S) < (3 - \frac{1}{m}) C^*_{\max}$.*

Proof. Assume job j with $C_j(S) = C_{\max}(S)$. As S is a nonpreemptive LRF schedule, there are at least $\frac{m+1}{2}$ machines busy during all time instances before time $t_j(0) = C_j(S) - p_j$. This leads to $C^*_{\max} \geq \frac{1}{m}\sum_{j\in\tau} p_j m_j > \frac{1}{m}\frac{(m+1)(C_j(S)-p_j)}{2}$.

Therefore, we have $C^*_{\max} \geq p_j$ and $C^*_{\max} > \frac{m+1}{2m}(C_{\max}(S) - p_j) \geq \frac{1}{2-\frac{1}{m}}(C_{\max}(S) - p_j)$ which results in

$$C_{\max}(S) = C_{\max}(S) - p_j + p_j < \left(3 - \frac{1}{m}\right) C^*_{\max}. \qquad \square$$

If $m_j = 1$ holds for all jobs $j \in \tau$, the LRF schedule S becomes a list schedule and therefore guarantees $C_{\max}(S) \leq (2 - \frac{1}{m}) C^*_{\max}$.

5. The algorithm. In this section, we introduce the new preemptive algorithm *PSRS* (preemptive Smith ratio scheduling), shown in Table 1.

Let us start the discussion of Algorithm PSRS by explaining the elements and components used during its execution. Q is the list of those jobs that are not yet scheduled. The order of Q is determined by the modified Smith ratio (largest ratio first). t_s is a local variable of the algorithm and gives the earliest starting time of the next job to be scheduled. Note that t_s cannot decrease during the run of the algorithm and that it is not smaller than the starting time of the job that has been scheduled last. Therefore, all jobs will start in the order given by Q. Next, $T(m')$ is a function returning the first time instance after t_s such that at least m' machines are

TABLE 1
The preemptive Algorithm PSRS.

```
Create a priority list Q for all jobs such that job i precedes job j if s_i > s_j;
t_s = 0;
while (Q ≠ ∅){
        pick the next job j and delete it from Q;
        d_j = 0;
        if (m_j > m/2 and p_j/v ≤ T(m_j) − T(m/2)) {
                t_j(0) = T(m/2) + p/2 + p_j/v;
                t_s = t_j(0) + p_j + p/2;
                for all jobs i with t_i(2d_i + 1) ≥ t_j(0) − p/2 do {
                        d_i = d_i + 1;
                        t_i(2d_i + 1) = t_i(2d_i − 1) + p_j + p;
                        t_i(2d_i − 1) = t_i(0);
                        t_i(2d_i) = t_s − p/2;
                }
        } else {
                t_j(0) = T(m_j);
                t_s = t_j(0);
        }
        t_j(1) = t_j(0) + p_j;
}
```

available in the temporary schedule at time t_s. As no job in this temporary schedule starts after time t_s, we can formally write

$$T(m') = \min \left\{ t | t \geq t_s \text{ and } m - m' \geq \sum_j m_j \text{ with } t_j(2d_j + 1) > t \right\}.$$

Note that $t_j(2d_j+1)$ is a value that may change during the further run of the algorithm and therefore may be different from the final completion time $C_j(S)$. Finally, $v \leq 1$ is a constant that is used to minimize the approximation factor of Algorithm PSRS. The value of v will be determined in the next section.

Intuitively, Algorithm PSRS removes the first job j from Q and schedules it in a nonpreemptive (LRF) fashion as long as the job needs at most 50% of all machines ($m_j \leq \frac{m}{2}$). However, if a job j requires more than $\frac{m}{2}$ machines, we determine the difference between the first time instances after t_s where $\frac{m}{2}$ and m_j machines are available, respectively. If this time difference is less than the ratio $\frac{p_j}{v}$, we schedule j in the same way as those jobs with less parallelism; that is, j starts at time $T(m_j)$ and runs to completion. Otherwise at time $T(\frac{m}{2}) + \frac{p_j}{v}$, we preempt all jobs in the temporary schedule that do not finish before that time and start job j at time $T(\frac{m}{2}) + \frac{p_j}{v} + \frac{p}{2}$ as we need to account for the preemption penalty to interrupt those other jobs. Note that we also preempt jobs that would complete at time $T(\frac{m}{2}) + \frac{p_j}{v}$. Those jobs complete immediately after their execution is resumed. This is only done for the sake of a slightly simpler analysis. After job j has been completed, the execution of those preempted jobs is resumed, thus causing another preemption penalty of $\frac{p}{2}$. The next job cannot start before the execution of the preempted jobs is resumed. This assures that the necessary machines are available to resume execution of the preempted job without migration. Therefore, any preemption-causing job will not execute concurrently with any other job. Finally, note that jobs $j \in \tau$ with $m_j > \frac{m}{2}$ are not preempted.

In further parts of this paper, we use the following notation for variables or functions that change their values during an iteration of the algorithm: $q(j)$ and $\overline{q}(j)$

denote the value of a variable or a function q at the beginning and at the end of the iteration where job j is scheduled, respectively.

Now, we discuss the validity of the schedule produced by PSRS.

LEMMA 5.1. *Consider a job system τ and a machine system $\mathcal{M}$. Then PSRS always generates a valid schedule that does not require migration.*

Proof. First note that at the beginning of any iteration, $t_j(2d_j) \leq t_s$ holds for any job j that is already scheduled ($j \in \tau \backslash Q$). Initially, PSRS assigns to each job j an integer $d_j = 0$ and time instances $t_j(0)$ and $t_j(1)$ with $t_j(1) - t_j(0) = p_j \geq 1$ and $t_s(j) \leq t_j(0) \leq \overline{t_s}(j)$. During a later iteration, job j' may cause preemption and $d_j(j')$ may become $\overline{d_j}(j') = d_j(j') + 1$ while the schedule remains unchanged in the interval $[0, 2d_j(j'))$. The validity of the scheduling conditions for $t_j(2\overline{d_j}(j') - 1)$, $t_j(2\overline{d_j}(j'))$, and $t_j(2\overline{d_j}(j') + 1)$ can easily be verified. Finally, as already mentioned above, each preempted job can be continued on the same set of machines it has used before. ∎

The time complexity of PSRS is $O(|\tau|^2)$ as in the worst case, $O(|\tau|)$ jobs may preempt $O(|\tau|)$ jobs each. Also note that the LRF schedules for sequential jobs and for jobs requiring at most $\frac{m}{2}$ machines are both included in Algorithm PSRS. Therefore, if the job system τ observes the corresponding resource restrictions, PSRS guarantees $C(S) < \frac{\sqrt{2}+1}{2} C^*$ [8] and $C(S) < 2C^*$, respectively.

6. Approximation factors. In this section, we address the approximation factors for Algorithm PSRS. First, we derive the makespan approximation factor.

THEOREM 6.1. *$C_{\max}(S) < (2 + \frac{1}{v} + p) C^*_{\max}$ holds for all PSRS schedules S.*

Proof. First, we address the average machine usage in the time interval $[0, (\overline{T(\frac{m}{2})})(i))$ with i being the job scheduled last by PSRS. Note that $C_{\max}(S) - (\overline{T(\frac{m}{2})})(i) \leq \max_{j \in \tau}\{p_j\}$ holds.

Consider the scheduling of job j. Algorithm PSRS will not introduce any additional idle time in the time interval $[0, (\overline{T(\frac{m}{2})})(j))$ during the later scheduling of any other job.

If $m_j \leq \frac{m}{2}$ holds, then more than $\frac{m}{2}$ machines are used at any time instance in $[(T(\frac{m}{2}))(j), (\overline{T(\frac{m}{2})})(j))$.

Assume job j with $m_j > \frac{m}{2}$. Then fewer than m_j machines are idle at any moment in time interval $[(T(\frac{m}{2}))(j), (T(m_j))(j))$. If j does not cause any preemption, then $(T(m_j))(j) - (T(\frac{m}{2}))(j) < \frac{p_j}{v}$ and $(\overline{T(\frac{m}{2})})(j) = (T(m_j))(j) + p_j$ hold. At least m_j machines are used in S at any time instance of the interval $[(T(m_j))(j), (\overline{T(\frac{m}{2})})(j))$. Therefore, at least the fraction $\frac{1}{\frac{1}{v}+1} \leq \frac{1}{2}$ of all machines is used on average in the time frame $[(T(\frac{m}{2}))(j), (\overline{T(\frac{m}{2})})(j))$ of schedule S.

Next, assume that job j preempts jobs $j_1, \ldots, j_k$. Therefore, we have $m_j + \sum_{\mu=1}^{k} m_{j_\mu} > m$. Execution of jobs $j_1, \ldots, j_k$ is resumed at time $t_j(1) + \frac{p}{2} = (T(\frac{m}{2}))(j) + (1 + \frac{1}{v})p_j + p = (\overline{T(\frac{m}{2})})(j)$. During the time frame $[(T(\frac{m}{2}))(j), (\overline{T(\frac{m}{2})})(j))$, the total used machine-time product is at least $\sum_{\mu=1}^{k} m_{j_\mu} \frac{1}{v} p_j + m_j p_j > m p_j$. With $(\overline{T(\frac{m}{2})})(j) - (T(\frac{m}{2}))(j) = p_j(\frac{1}{v} + 1) + p$ and $p_{j'} \geq 1$ for all jobs $j' \in \tau$, we can therefore conclude that more than the fraction $\frac{1}{\frac{1}{v}+1+p}$ of all machines is used on average in $[(T(\frac{m}{2}))(j), (\overline{T(\frac{m}{2})})(j))$ and in $[0, (\overline{T(\frac{m}{2})})(i))$ as well.

Further, we have $C^*_{\max} \geq \sum_{j \in \tau} \frac{m_j p_j}{m}$ and $C^*_{\max} \geq \max_{j \in \tau} p_j$. This results in

$$C_{\max}(S) = C_{\max}(S) - \left(\overline{T\left(\frac{m}{2}\right)}\right)(i) + \left(\overline{T\left(\frac{m}{2}\right)}\right)(i)$$

TABLE 2
Worst-case job system for makespan schedules.

p_j	m_j	w_j	Number of jobs
$\frac{1}{v}$	1	p_j	$2m$
1	$m-1$	m_j	m
m	1	p_j	1

TABLE 3
Worst-case job system for weighted completion time schedules.

p_j	m_j	w_j	Number of jobs
$\frac{1}{v}$	1	p_j	$k(x+1)$
1	$m-x$	m_j	k
ky	1	p_j	x
1	m	0	$\lfloor kyv \rfloor$

$$< \max_{j\in\tau} p_j + \left(\frac{1}{v} + 1 + p\right) \frac{1}{m} \sum_{j\in\tau} m_j p_j$$
$$\leq C^*_{\max} + \left(\frac{1}{v} + 1 + p\right) C^*_{\max} = \left(\frac{1}{v} + 2 + p\right) C^*_{\max}. \qquad \square$$

Note that the approximation factor is a function of v and p. This factor is tight for Algorithm PSRS and large values of m. This can be seen from the job system in Table 2 for which PSRS produces $C_{\max}(S) = (2 + \frac{1}{v} + p)C^*_{\max}$ if $m \to \infty$ holds. As the modified Smith ratio is the same for all jobs, we assume that in list Q of Algorithm PSRS, two jobs of the first group are always followed by one job of the second group, while the single job of the third group is the last job of Q. This results in $C_{\max}(S) = m(\frac{1}{v} + 1 + p) + m$. In an optimal schedule, the job of the last group is scheduled first. Then all jobs of the second group are executed concurrently with this job followed by all jobs of the first group. This leads to $C^*_{\max} = m + \frac{2}{v}$.

Next, we introduce the job systems of Table 3. Those job systems consist of four different types of jobs. Later, we will prove that one of those job systems produces the maximum ratio $\frac{C(S)}{C^*}$ of any PSRS schedule S for $k \gg m \to \infty$. First, we determine $C(S)$ and C^*, respectively. The real number $y \geq \frac{1}{k}$ and the nonnegative integer $x \leq \frac{m}{2} - 1$ in Table 3 are parameters.

The modified Smith ratio is 1 for all jobs of the first three types. Therefore, any order of those jobs in list Q is possible. We assume that $x + 1$ jobs of the first type are always followed by one job of the second type. The type 2 job causes preemption of the previous $x + 1$ type 1 jobs. All jobs of the third type are put into Q after all jobs of the first two types. Note that the last type of job has a modified Smith ratio of 0. Hence, these jobs must be at the end of Q and they do not contribute to C^*. However, these jobs delay the completion of type 3 jobs and therefore indirectly increase $C(S)$ of the PSRS schedule S.

A schedule with an optimal total weighted completion time schedule for this job system is obtained by starting all type 3 jobs at time 0 (contribution of those jobs to

the total weighted completion time: xk^2y^2). All type 2 jobs are executed one after another. The whole staple of these jobs also starts at time 0 (contribution of all type 2 jobs: $(m-x)\frac{k(k+1)}{2}$). Finally, all jobs of type 1 are scheduled at the earliest time possible, that is, on top of either the type 2 jobs or the type 3 jobs. This leads to three possible cases:

1. All type 1 jobs do not complete after time k, that is, the completion time of the last type 2 job.
2. All type 1 jobs do not complete after time ky, that is, the completion time of all type 3 jobs.
3. Some type 1 jobs complete after $\max\{k, ky\}$.

By comparison of those three cases, it can be determined that the worst ratio $\frac{C(S)}{C^*}$ is obtained for case 2, that is, if $y > 1 + \frac{x+1}{v(m-x)}$ holds. Therefore, the contribution of type 1 jobs is between $\frac{k(x+1)}{v}k + \frac{m-x}{2v^2}\lfloor\frac{k(x+1)}{m-x}\rfloor(\lfloor\frac{k(x+1)}{m-x}\rfloor + 1)$ and $\frac{k(x+1)}{v}k + \frac{m-x}{2v^2}\lceil\frac{k(x+1)}{m-x}\rceil(\lceil\frac{k(x+1)}{m-x}\rceil + 1)$.

In the PSRS schedule, the first $x+1$ type 1 jobs and the first job of type 2 will contribute to the total weighted completion time with $(\frac{x+1}{v}+m-x)(\frac{1}{v}+1+p)-\frac{(m-x)p}{2}$. Therefore, the contribution of all type 1 and type 2 jobs together is $\frac{k(k+1)}{2}(\frac{x+1}{v}+m-x)(\frac{1}{v}+1+p) - k\frac{(m-x)p}{2}$. Finally, all type 3 jobs are executed after all type 1 and type 2 jobs. As already mentioned, the completion time of those jobs is increased by type 4 jobs. This results in a contribution of not more than $kxy(kyv+k)(\frac{1}{v}+1+p)$. Hence, the high order terms of $C(S)$ and C^* are given by

$$\lim_{k\gg m, m\to\infty} C^* = k^2\left(xy^2 + \frac{1}{2}(m-x)\left(\frac{x+1}{v(m-x)}+1\right)^2\right) \quad \text{and}$$

$$\lim_{k\gg m, m\to\infty} C(S) = k^2\left(xy(yv+1) + \frac{1}{2}\left(\frac{x+1}{v}+m-x\right)\right)\left(\frac{1}{v}+1+p\right).$$

The maximum ratio $\frac{C(S)}{C^*}$ of all PSRS schedules S for the job system of Table 3 with $k \gg m \to \infty$ depends on p and is denoted as $f_C(p)$. It is obtained by solving the following optimization problem:

$$f_C(p) = \lim_{k,m\to\infty} \min_{0<v\le 1}\left(\max_{y\ge 0, 0\le x<\frac{m}{2}}\left(\frac{C(S)}{C^*}\right)\right).$$

First, we discuss the reason to assume $m \to \infty$. There are $k(x+1)$ type 1 jobs in the job systems of Table 3. If we allow any value between 0 and $\frac{m}{2}$ for x in the optimization problem, then we need to consider only the contribution of the additional k type 1 jobs as their influence on the ratio of $\frac{C(S)}{C^*}$ varies with the size of m. The contribution of those jobs to C^* is at least $\frac{k^2}{v}$, while they contribute $\frac{k(k+1)}{2v}(\frac{1}{v}+1+p)$ to $C(S)$. As we have $\frac{k(k+1)}{2} \le k^2$ for $k \ge 1$, we can ignore this contribution in order to determine an upper bound of the worst case; that is, we can assume $m \to \infty$. With respect to Algorithm PSRS, this means that we allow a parallel job j to preempt jobs that together use $m - m_j$ instead of $m - m_j + 1$ machines. In particular, this includes the case where $m_j = \lceil\frac{m}{2}\rceil$ holds; see the proof of Corollary 6.5, step 4. Clearly, this modification does not reduce $C(S)$. However, note that it is used for the analysis only.

Before determining the approximation factor for the total weighted completion time, we introduce a lower bound $\tilde{C}^*$ for the optimal total weighted completion time

C^*. To obtain $\tilde{C}^*$, we simply assume that for all jobs, we have either $m_j = 1$ or $m_j > \lceil \frac{m}{2} \rceil$ in the optimal schedule. This is achieved by modifying the job system τ such that every job $j \in \tau$ with $1 < m_j \leq \lceil \frac{m}{2} \rceil$ is replaced by m_j sequential jobs $i_1, \ldots, i_{m_j}$ with $p_{i_\kappa} = p_j$, $m_{i_\kappa} = 1$, and $s_{i_\kappa} = s_j$ for $1 \leq \kappa \leq m_j$. Any valid schedule for τ is valid for the new job system as well. Therefore, $\tilde{C}^*$ is a lower bound for C^*. Note that $\tilde{C}^* = C^*$ holds for the job systems of Table 3.

If there are no jobs $j \in \tau$ with $2 \leq m_j \leq \lceil \frac{m}{2} \rceil$ and all jobs of job system τ have a modified Smith ratio of 1, then there are no intermittent idle times on any machine in the optimal schedule as no parallel jobs ($m_j > \frac{m}{2}$) can be executed concurrently. Note that the optimal schedule is nonpreemptive in this case and the total weighted completion time for each machine of the system is $\frac{1}{2}((\sum_j p_j)^2 + \sum_j p_j^2)$ with the sums being taken over all jobs $j \in \tau$ that are scheduled on this machine [12].

Under these conditions, the total weighted completion time is minimized if the makespan is the same for all machines of the system due to reasons of convexity [8]. Although the same makespan for all machines cannot be achieved for the job systems of Table 3, we will use this property for a subset of machines. In particular, we assume that in the optimal schedule, all those machines executing more than one sequential job have the same makespan. Again for the job systems of Table 3, this assumption does not result in a deviation from C^* as for $k \to \infty$, the execution time of all type 1 and type 2 jobs is small compared to the makespan of the machines assigned to them.

Further, for our analysis we increase the completion time of each preemption-causing job in a PSRS schedule by $\frac{p}{2}$; that is, the job completes at the same time when the execution of the preempted jobs is resumed. Hence for such a preemption-causing job j, we have $t_j(1) = t_s(j) = (\overline{T(\frac{m}{2})})(j)$. This modification increases the ratio $\frac{C(S)}{\tilde{C}^*}$ while $f_C(p)$ remains unchanged.

The next theorem is one of the main results of this paper and states that the approximation factor of PSRS is $f_C(p)$.

THEOREM 6.2. *$\frac{C(S)}{C^*} \leq f_C(p)$ holds for all PSRS schedules S.*

To prove Theorem 6.2, we gradually restrict the number of job systems, which must be considered in order to generate the maximum deviation of $C(S)$ from the lower bound $\tilde{C}^*$ of the optimum C^*, until we reach the job systems of Table 3. To enhance readability, the proof of Theorem 6.2 is divided into a lemma and several corollaries.

The total weighted completion times of the PSRS schedule and the optimal schedule for the job systems of Table 3 are determined only by jobs with the same modified Smith ratio. Therefore, we start our proof by showing that for the determination of the approximation factor of Algorithm PSRS, it is sufficient to consider only the completion times of the jobs with maximum modified Smith ratio in all job systems and PSRS schedules. To this end, we introduce the following notations:

$$\hat{\tau} = \left\{ i | i \in \tau \text{ and } s_i = \max_{j \in \tau}(s_j) \right\},$$
$$\hat{C}(S) = \sum_{i \in \hat{\tau}} w_i t_i (2d_i + 1).$$

$\hat{C}(S)$ is a restriction of $C(S)$ to the contribution of all jobs of the subset $\hat{\tau}$ in schedule S. As the order of all jobs $j \in \hat{\tau}$ in Q is arbitrary, there may be several different PSRS schedules S with different values of $\hat{C}(S)$ for a given job system τ.

Further, we consider $\tilde{C}^*(\hat{\tau})$, which is the lower bound of the optimal total weighted completion time for a job system consisting of only the jobs in $\hat{\tau}$.

With these notations, we can state the next lemma, which is closely related to Theorem 2 of Kawaguchi and Kyan's paper [8].

LEMMA 6.3 (Kawaguchi and Kyan). *If* $\frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})} \leq \lambda$ *holds for all job systems and PSRS schedules* S*, then* $\frac{C(S)}{\tilde{C}^*}$ *is also upper bounded by* λ.

This lemma allows us to assume that for the purpose of worst-case analysis, any job system consists only of some jobs with a modified Smith ratio of 1 while $s_j = 0$ holds for all other jobs j of the job system as s_j is only used in Algorithm PSRS to determine the order of Q. Note that this property is true for the job systems of Table 3. Although the proof can be directly derived from [8], it is restated here for the sake of completeness.

Proof. The proof is done by induction over the number k of different modified Smith ratios in a job system. The lemma clearly holds for $k = 1$.

Assume that the claim holds for all job systems with k different modified Smith ratios and that τ is a job system with $k+1$ different modified Smith ratios. We use $s_1 = \max_{j\in\tau}\{s_j\}$ and $s_2 = \max_{j\in\tau\setminus\hat{\tau}}\{s_j\}$ to denote the largest and the second largest modified Smith ratio values of τ, respectively. Further, we have $\frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})} \leq \lambda$.

We generate a job system τ' by modifying the weights of the jobs in τ as follows:

$$w_j \leftarrow \begin{cases} \frac{s_2}{s_1} w_j & \text{for } j \in \hat{\tau}, \\ w_j & \text{for } j \in \tau\setminus\hat{\tau}, \end{cases}$$

Note that any PSRS schedule $S(\tau)$ is also a PSRS schedule for τ' as any order of jobs in τ also can be applied to τ'. Based on our induction assumption, $\frac{C(S(\tau'))}{\tilde{C}^*(\tau')} \leq \lambda$ holds for such a schedule $S(\tau')$ as there are only k different modified Smith ratios in τ'.

As the difference between the total weighted completion times of schedule S for job systems τ and τ' is caused only by jobs in $\hat{\tau}$, we have

$$C(S(\tau)) - C(S(\tau')) = \left(1 - \frac{s_2}{s_1}\right)\hat{C}(S).$$

Further, note that the contribution of all jobs in $\hat{\tau}$ to $\tilde{C}^*(\tau)$ is at least $\tilde{C}^*(\hat{\tau})$. Therefore, we can state the following relationship between $\tilde{C}^*(\tau)$ and $\tilde{C}^*(\tau')$:

$$\tilde{C}^*(\tau) \geq \left(1 - \frac{s_2}{s_1}\right)\tilde{C}^*(\hat{\tau}) + \tilde{C}^*(\tau').$$

Combining those inequalities finally results in

$$\frac{C(S(\tau))}{\tilde{C}^*(\tau)} \leq \frac{C(S(\tau')) + (1 - \frac{s_2}{s_1})\hat{C}(S)}{\tilde{C}^*(\tau') + (1 - \frac{s_2}{s_1})\tilde{C}^*(\hat{\tau})} \leq \lambda \frac{\tilde{C}^*(\tau') + (1 - \frac{s_2}{s_1})\tilde{C}^*(\hat{\tau})}{\tilde{C}^*(\tau') + (1 - \frac{s_2}{s_1})\tilde{C}^*(\hat{\tau})} = \lambda. \qquad \square$$

However, jobs in $\tau\setminus\hat{\tau}$ may cause preemption and therefore indirectly affect $\hat{C}(S)$ by increasing the completion time of some jobs in $\hat{\tau}$; see, for instance, the type 4 jobs in the job systems of Table 3. As we are interested in this property only for jobs $j \in \tau\setminus\hat{\tau}$, we assume that $m_j = m$ holds for those jobs. To formally describe an upper bound for the influence of those jobs, we define the time t_b in a PSRS schedule to be

$$t_b = \min_{j\in\tau\setminus\hat{\tau}} \left\{ T\left(\frac{m}{2}\right)(j) \right\}.$$

t_b is a lower bound for the first time in a PSRS schedule at which a job $j \in \tau\backslash\hat{\tau}$ with $m_j = m$ may be scheduled. If $\tau = \hat{\tau}$ holds, then we simply add a job j with $w_j = 0$, $m_j = m$, and $p_j = 1$. This job cannot decrease $\hat{C}(S)$ while $\tilde{C}^*$ remains unchanged.

Therefore, $t_j(0) \leq t_b$ holds for all $j \in \hat{\tau}$, and any job $i \in \hat{\tau}$ with $t_i(2d_i + 1) < t_b + \frac{1}{v}$ is not preempted by any job $j \in \tau\backslash\hat{\tau}$ as we have $p_j \geq 1$. We use the set $\hat{\tau}_t = \{j \in \hat{\tau} | t_j(2d_j + 1) \geq t_b + \frac{1}{v}\}$ to describe those jobs whose completion time is delayed by jobs in $\tau\backslash\hat{\tau}$. Let $\check{p}_j$ be the remaining processing time of a job $j \in \hat{\tau}$ at time t_b in the PSRS schedule S. Then there can be at most $\lfloor v\check{p}_j \rfloor$ jobs with processing time 1 and a modified Smith ratio of 0 that preempt job j. Therefore, we have $t_j(2d_j + 1) \leq t_b + \check{p}_j(1 + v + vp)$. As we are interested only in the worst case, it is assumed from now on that $t_j(2d_j + 1) = t_b + \check{p}_j(1 + v + vp)$ holds for all jobs $j \in \hat{\tau}_t$. This allows us to ignore all jobs $j \in \tau\backslash\hat{\tau}$. Again note that the following properties hold in the job systems of Table 3:

(i) $t_b = k(\frac{1}{v} + 1 + p)$;
(ii) $\hat{\tau}_t$ consists of all type 3 jobs with $\check{p}_j = p_j$;
(iii) $t_j(2d_j + 1) = t_b + p_j(1 + v + vp)$ for all jobs $j \in \hat{\tau}_t$ and $k \to \infty$.

The rest of the proof of Theorem 6.2 is divided into three corollaries. As already mentioned, we repeatedly transform a job system τ into a new (more restricted) job system τ'. To distinguish between corresponding expressions of both systems we use the single quote character, for instance, schedule $S(\tau)$ for job system τ and another schedule $S'(\tau')$ for job system τ'.

Assume that a job system τ with $\frac{\hat{C}(S(\tau))}{\tilde{C}^*(\hat{\tau})} \geq \Delta > 0$ is transformed into a job system τ' such that $\hat{C}(S(\tau)) - \hat{C}(S'(\tau')) \leq \Delta(\tilde{C}^*(\hat{\tau}) - \tilde{C}^*(\hat{\tau}'))$ holds. Then we have

$$\frac{\hat{C}(S')}{\tilde{C}^*(\hat{\tau}')} = \frac{\hat{C}(S) - (\hat{C}(S) - \hat{C}(S'))}{\tilde{C}^*(\hat{\tau}')} \geq \frac{\hat{C}(S) - \Delta(\tilde{C}^*(\hat{\tau}) - \tilde{C}^*(\hat{\tau}'))}{\tilde{C}^*(\hat{\tau}')}$$
$$\geq \frac{\hat{C}(S) - \frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})}(\tilde{C}^*(\hat{\tau}) - \tilde{C}^*(\hat{\tau}'))}{\tilde{C}^*(\hat{\tau}')} = \frac{\hat{C}(S)\frac{\tilde{C}^*(\hat{\tau}')}{\tilde{C}^*(\hat{\tau})}}{\tilde{C}^*(\hat{\tau}')} = \frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})}.$$

For worst-case analysis, it is therefore sufficient to consider only job system τ' instead of both systems τ and τ'. We will repeatedly use this approach to restrict the number of those job systems which may generate a worst case.

In the following, we say that all jobs $j \in \tau$ with $m_j > \frac{m}{2}$ are *wide* jobs. In the first of the above mentioned three corollaries, we address those wide jobs.

COROLLARY 6.4. *Assume a job system τ with $\frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})} \geq 1 + \frac{1}{v} + p$ and a PSRS schedule $S(\tau)$. Then there is a job system τ' and a PSRS schedule $S'(\tau')$ with $\frac{\hat{C}(S'(\tau'))}{\tilde{C}^*(\hat{\tau}')} \geq \frac{\hat{C}(S(\tau))}{\tilde{C}^*(\hat{\tau})}$ such that for every job $j \in \hat{\tau}'$ with $m_j > \frac{m}{2}$*

1. *either we have $p_j = 1$ and j causes preemption in S',*
2. *or $(T(m_j))(j) = (T(\frac{m}{2}))(j)$ holds in Algorithm PSRS.*

Proof. We prove this corollary by using three different simple transformations that are described below.

1. *Time split of a wide job causing preemption.* A time split is a division of the job into several independent jobs with the same amount of parallelism. Assume a wide job $j \in \hat{\tau}$ with $p_j \geq 2$ that causes preemption in S. Then we have $t_j(1) = (T(\frac{m}{2}))(j) + (\frac{1}{v} + 1)p_j + p$. Remember that the completion time of job j has been increased by $\frac{p}{2}$.

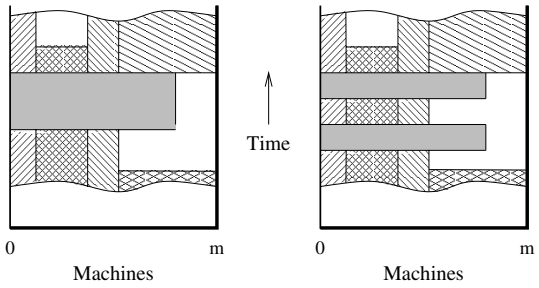


FIG. 3. *Time split of a job causing preemption ($p = 0$).*

We transform τ into τ' by replacing j in Q with two jobs j_1 and j_2 such that

$$m_{j_1} = m_{j_2} = m_j, \;\; s_{j_1} = s_{j_2} = s_j = 1, \;\; p_{j_1} = 1, \;\; \text{and} \;\; p_{j_2} = p_j - 1 \;\; \text{holds.}$$

S' is the result of applying Algorithm PSRS on job system τ' with the new list Q'. Note that S and S' are identical until time $(T(\frac{m}{2}))(j)$ and that both jobs j_1 and j_2 will cause preemption in S'. The completion times of jobs j_1 and j_2 in S' are given below:

$$t'_{j_1}(1) = \left(T'\left(\frac{m}{2}\right)\right)(j_1) + \frac{1}{v} + 1 + p = \left(T\left(\frac{m}{2}\right)\right)(j) + \frac{1}{v} + 1 + p$$

and

$$t'_{j_2}(1) = t_j(1) + p.$$

Further, no job in $\tau \cap \tau'$ can complete earlier in S' than it does in S; that is, $t_i(2d_i + 1) \le t'_i(2d'_i + 1)$ holds for all jobs $i \in \tau \cap \tau'$ (see also Figure 3). This results in

$$\begin{aligned}\hat{C}(S) - \hat{C}(S') &\le m_j p_j t_j(1) - m_{j_1} p_{j_1} t'_{j_1}(1) - m_{j_2} p_{j_2} t'_{j_2}(1) \\ &= m_j(p_j - 1)\left(\frac{1}{v} + 1 - p\right) \le m_j(p_j - 1)\left(\frac{1}{v} + 1 + p\right).\end{aligned}$$

It is easy to see that a valid schedule for τ' can be obtained from any valid schedule for τ by replacing job j with jobs j_1 and j_2. With $\check{t}_j$ denoting the completion time of job j in the optimal schedule for τ, we have

$$\tilde{C}^*(\hat{\tau}) - \tilde{C}^*(\hat{\tau'}) \ge m_j p_j \check{t}_j - m_{j_2} p_{j_2} \check{t}_j - m_{j_1} p_{j_1}(\check{t}_j - p_{j_2}) = m_j(p_j - 1).$$

Together with the remarks above, this yields

$$\frac{\hat{C}(S')}{\tilde{C}^*(\hat{\tau'})} \ge \frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})} \ge 1 + \frac{1}{v} + p.$$

2. *Scaling.* Next, a job system τ is transformed into τ'' by replacing each job $i \in \tau$ with job $j \in \tau''$ such that we have

$$m_j = m_i, \;\; p_j = a p_i, \;\; \text{and} \;\; w_j = a w_i$$

using a positive integer a. This results in $\frac{\hat{C}(S'')}{\tilde{C}^*(\hat{\tau''})} = \frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})}$ if $p = 0$ holds. Then we obtain τ' from τ'' by repeatedly applying the previous time splitting step to all jobs

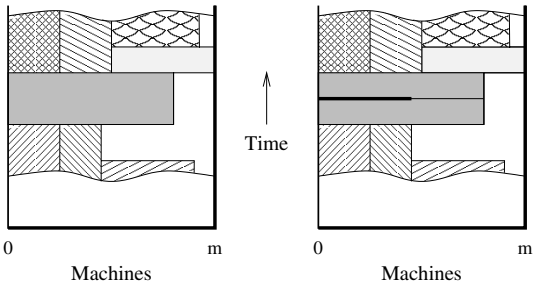


FIG. 4. *Time split of a wide job not causing preemption ($p = 0$).*

$j \in \tau''$ if those jobs cause preemption and have $p_j \geq 2$. Note that if a job $i \in \hat{\tau}$ is preempted k-times in schedule S, then the corresponding job $j \in \hat{\tau}'$ is preempted at least ak-times in S'. Also remember that $t_i(2d_i + 1) = t_b + \check{p}_i(1 + v + vp)$ holds for all jobs $i \in \hat{\tau}_t$. The same is true for all jobs $i \in \hat{\tau}'_t$. This scaling procedure will therefore result in $\hat{C}(S') \geq a\hat{C}(S)$, $\tilde{C}^*(\hat{\tau}') = a\tilde{C}^*(\hat{\tau})$, and in

$$\frac{\hat{C}(S')}{\tilde{C}^*(\hat{\tau}')} \geq \frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})} \geq 1 + \frac{1}{v} + p,$$

even if p is positive.

If all p_i are rational numbers, then a can be selected such that $p_j = 1$ holds for all jobs $j \in \hat{\tau}'$ that cause preemption in S'. As scaling will not decrease the ratio $\frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})}$, we can assume $k \to \infty$ and that $p_j = 1$ holds for every preemption-causing job j. These properties are valid for the job systems of Table 3.

3. *Time split of a wide job not causing preemption.* Assume a job $i \in \hat{\tau}$ with $m_i > \frac{m}{2}$ such that we have $(T(m_i))(i) - (T(\frac{m}{2}))(i) > 0$ and $\frac{p_i}{v} > (T(m_i))(i) - (T(\frac{m}{2}))(i)$ in Algorithm PSRS; that is, i does not cause preemption in schedule S. Now, job system τ is transformed into τ' by replacing i in Q with two jobs i_1 and i_2 such that $m_{i_1} = m_{i_2} = m_i$, $s_{i_1} = s_{i_2} = s_i = 1$, $p_{i_1} = v((T(m_i))(i) - (T(\frac{m}{2}))(i))$, and $p_{i_2} = p_i - p_{i_1}$ hold.

If we have $p_{i_1} < 1$ or $p_{i_2} < 1$, then the job system is appropriately scaled as described above. In S', job i_1 now causes preemption while $(T'(m_{i_2}))(i_2) = (T'(\frac{m}{2}))(i_2)$ holds for job i_2 in Algorithm PSRS; see Figure 4. Therefore, we have $t'_{i_2}(1) = t_i(1) + p$ and $t'_{i_1}(1) = t_i(1) - p_{i_2} + p$.

This results in

$$\begin{aligned}\hat{C}(S) - \hat{C}(S') &\leq m_i p_i t_i(1) - m_{i_1} p_{i_1} t'_{i_1}(1) - m_{i_2} p_{i_2} t'_{i_2}(1) \\ &= m_i(p_{i_1} p_{i_2} - p_{i_1} p - p_{i_2} p) \leq m_i p_{i_1} p_{i_2} \left(\frac{1}{v} + 1 + p\right) \quad \text{and} \\ \tilde{C}^*(\hat{\tau}) - \tilde{C}^*(\hat{\tau}') &\geq m_i p_{i_1} p_{i_2}. \qquad \square\end{aligned}$$

Note that Corollary 6.4 is valid for any p.

Now, we describe the properties of our next target job system. There are only wide and sequential jobs in this job system. All jobs in $\hat{\tau}_t$ start at the same time after the other jobs in $\hat{\tau}$ have completed. All other sequential jobs ($m_j = 1$) in $\hat{\tau}$ are preempted exactly once and have a remaining processing time of 0 when their execution is resumed. Finally, each wide job $j \in \hat{\tau}$ preempts $m - m_j$ sequential jobs;

that is, as many machines as possible are left idle. Again, all these properties are valid for the job systems of Table 3 if those additional k type 1 jobs are ignored in the PSRS schedule as discussed before.

COROLLARY 6.5. *Assume a job system τ with $\frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})} \geq 1 + \frac{1}{v} + p$ and a PSRS schedule S. Then there is another job system τ' and a PSRS schedule S' such that $\frac{\hat{C}(S')}{\tilde{C}^*(\hat{\tau}')} \geq \frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})}$ and the following properties hold:*

(i) *$m_i \geq \lceil \frac{m}{2} \rceil$ or $m_i = 1$ for all jobs $i \in \hat{\tau}'$;*
(ii) *$t'_i(0) = t'_b$ for all jobs $i \in \hat{\tau}'_t$ in S';*
(iii) *$p_i = \frac{1}{v}$ and $d'_i = 1$ for all jobs $i \in \hat{\tau}' \backslash \hat{\tau}'_t$ with $m_i = 1$;*
(iv) *$m - m_j$ jobs are preempted by any job $j \in \hat{\tau}'$ with $m_j \geq \lceil \frac{m}{2} \rceil$ in S.*

Proof. Again we use several simple transformations to prove this corollary.

1. *Time split after preemption.* In this part, we apply time splitting to all jobs that are preempted and have more than 0 processing time left when their execution is resumed. To this end, let $i \in \hat{\tau}$ be a job with $m_i \leq \frac{m}{2}$ and $t_i(2d_i + 1) > t_i(2d_i)$ that is preempted at least once by a job in $\hat{\tau}$ in PSRS schedule S. Then schedule S and job system τ are transformed into schedule S' and job system τ' by replacing i in S with two jobs i_1 and i_2 such that $m_{i_1} = m_{i_2} = m_i$, $s_{i_1} = s_{i_2} = s_i = 1$, $p_{i_1} = t_i(1) - t_i(0)$, $p_{i_2} = p_i - p_{i_1}$, $d'_{i_1} = 1$, $d'_{i_2} = d_i - 1$, $t'_{i_1}(\mu) = t_i(\mu)$ for $0 \leq \mu \leq 2$, $t'_{i_1}(3) = t_i(2)$, and $t'_{i_2}(\mu - 2) = t_i(\mu)$ for $2 \leq \mu \leq 2d_i + 1$ hold.

Note that job i is split in S at the time when its execution is resumed after its first preemption. A temporary violation of the minimal execution time condition for job i_2 always can be removed by use of the scaling procedure of Corollary 6.4. This is also true for other parts of this proof.

For this time splitting of job i, the condition $t'_{i_2}(2d'_{i_2} + 1) - t'_{i_2}(0) \leq p_{i_2}(1 + v + vp)$ holds, and the completion time of no other job is affected. Therefore, we have

$$\begin{aligned}
\hat{C}(S) - \hat{C}(S') &\leq m_i p_i t_i(2d_i + 1) - m_{i_2} p_{i_2} t'_{i_2}(2d'_{i_2} + 1) - m_{i_1} p_{i_1} t'_{i_1}(2d'_{i_1} + 1) \\
&= m_i p_{i_1}(t'_{i_2}(2d'_{i_2} + 1) - t'_{i_2}(0)) \\
&\leq m_i p_{i_1} p_{i_2}(1 + v + vp) \\
&\leq m_i p_{i_1} p_{i_2} \left(1 + \frac{1}{v} + p\right) \quad \text{and} \\
\tilde{C}^*(\hat{\tau}) - \tilde{C}^*(\hat{\tau}') &\geq m_i p_{i_1} p_{i_2}.
\end{aligned}$$

To generate PSRS schedule S' for τ', we simply construct Q' from Q by replacing i with i_1 and introducing i_2 just after the job j, which causes preemption of i_1 in S'. This is possible as $s_j = 1$ holds.

The repeated application of this procedure in combination with the proper scaling will assure that each job i in $\hat{\tau}' \backslash \hat{\tau}'_t$ with $m_i \leq \frac{m}{2}$ is preempted at most once and that it will complete immediately when its execution is resumed. Further, no job in $\hat{\tau}_t$ is preempted by any job in $\hat{\tau}$.

2. *Machine split after a preemption.* The division of a parallel job into several independent jobs with the same processing time and the same modified Smith ratio is called a machine split of this job.

If a job $i \in \hat{\tau}$ starts in schedule S at time 0 or at the same time that a preemption-causing job completes, we replace i with m_i identical jobs j such that $m_j = 1$, $p_j = p_i$, and $s_i = s_j = 1$ hold. Note that we have $t_s(i) = t_i(0)$. Therefore, none of the m_i newly generated sequential jobs can start before $t_i(0)$ in PSRS schedule S'. Hence, this transformation does not affect the cost of the PSRS schedule and results in $\hat{C}(S) - \hat{C}(S') = 0 \leq \tilde{C}^*(\hat{\tau}) - \tilde{C}^*(\hat{\tau}')$.

3. *Removal of sequential jobs.* Next, we transform τ into τ' by removing a sequential job $i \in \hat{\tau} \backslash \hat{\tau}_t$, if this procedure does not decrease the completion time of any other job and if there is a (preemption-causing) wide job $j \in \hat{\tau}$ with $t_i(2d_i+1) \leq t_j(1)$. Note that after the removal of job i, a combined machine-time product of at least $\frac{mt_j(1)}{1+\frac{1}{v}+p}$ is still used for the execution of all jobs in $\hat{\tau} \backslash \hat{\tau}_t$ that complete no later than $t_j(1)$ in S'; see step 1 of this proof and the proof of Theorem 6.1. As already mentioned, we assume that the makespan is the same for all machines executing more than one sequential job in the schedule producing $\tilde{C}^*(\hat{\tau}')$. Therefore, this makespan is at least $\frac{t_j(1)}{1+\frac{1}{v}+p}$. With $k \to \infty$, this results in

$$\hat{C}(S) - \hat{C}(S') \leq p_i t_j(1)$$

and

$$\tilde{C}^*(\hat{\tau}) - \tilde{C}^*(\hat{\tau}') \geq \frac{1}{2m}\left(\frac{mt_j(1)}{1+\frac{1}{v}+p} + p_i\right)^2 - \frac{1}{2}\left(\frac{t_j(1)}{1+\frac{1}{v}+p}\right)^2 \geq p_i \frac{t_j(1)}{1+\frac{1}{v}+p}.$$

If a job $j \in \hat{\tau}$ preempts n sequential jobs, then we transform τ into τ' with the help of steps 1 and 3 by removing $n + m_j - m$ of these sequential jobs. This will not affect the completion time of any other job.

4. *Increase of the preemption ratio.* Next, we address those time intervals of the schedule where preemption occurs less frequently. Intuitively, it seems obvious that these intervals cannot contribute to a worst case as there the ratio of busy machines to idle machines is not worse than in intervals with more preemption. To prove this, we consider a time interval $T = [t_a, t_e)$ that starts either at 0 or at $t_{j_1}(1) < t_b$ with $j_1 \in \hat{\tau}$ being a preemption-causing job. Let j_2 be the first job in $\hat{\tau}$ that completes after time t_a in S. We define $t_e = t_{j_2}(2d_{j_2} + 1)$ and consider those cases where job j_2 does not cause preemption and cannot be removed using step 2.

Let $i_1, \ldots, i_n$ be the jobs which start at time t_a in schedule S. As shown in step 2 of this proof, we can assume that all these jobs are sequential. Note that there are more than $\frac{m}{2}$ of those jobs and all of them do not complete before time t_e.

Now, we perform several transformations with jobs $i_1, \ldots, i_n$ in order to introduce preemption in T. Remember that we can always apply the scaling procedure to avoid any problems with the minimum processing time.

(i) Every job i_μ with $t_{i_\mu}(1) > t_e$ is split at time t_e using the previously described method. The descendant of job i_μ that starts at time t_a replaces job i_μ in Q'. Now, all jobs that are executed in T start at time t_a and complete at time t_e. Note that the resulting schedule will probably not be a PSRS schedule.

(ii) We use scaling with a factor a and time splitting to obtain b groups of $\lfloor \frac{m}{2} \rfloor$ jobs with processing time $\frac{1}{v}$ and b groups of $\lceil \frac{m}{2} \rceil$ jobs with processing time 1 such that the combined processing time of all those jobs is equivalent to $a(t_e - t_a)n$. In schedule S'', we arrange those jobs such that all jobs of a group are executed concurrently and one group with processing time $\frac{1}{v}$ is immediately followed by one group with processing time 1. For the resulting job system τ'' and schedule S'', we have $\frac{\hat{C}(S'')}{\tilde{C}^*(\hat{\tau}'')} \geq \frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})}$.

(iii) Next, we combine each group with processing time 1 into a single wide job to generate job system τ'. Note that we still have $\tilde{C}^*(\hat{\tau}') = \tilde{C}^*(\hat{\tau}'')$.

(iv) Finally, we arrange Q' such that the original jobs, executing in T, are replaced by a repeated sequence of $\lfloor \frac{m}{2} \rfloor$ sequential jobs always followed by one wide

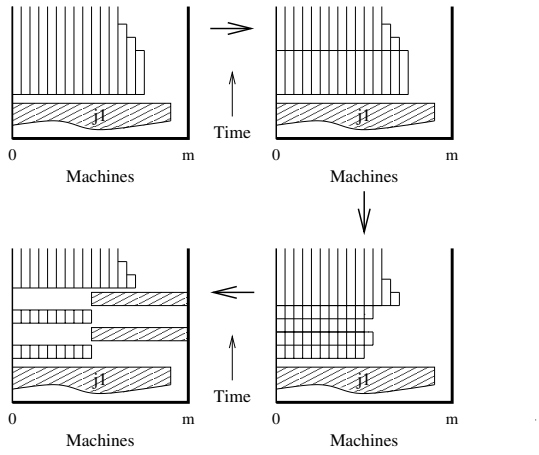


FIG. 5. *Increase of the preemption ratio.*

TABLE 4
Source (top) and target (bottom) job system for the special case.

p_i	m_i	s_i	Number of jobs
g	1	1	$\frac{m}{2}$
$b-a$	1	1	$\frac{m}{2}$
1	$\frac{m}{2}$	1	a

p_i	m_i	s_i	Number of jobs
$d-c$	1	1	m'
1	$m-m'$	1	$e-c$
1	m	1	c

job. These jobs are introduced into Q' before the other descendants of jobs $i_1, \ldots, i_n$. This produces PSRS schedule S'. The completion time of no job completing after at_e in schedule S'' is decreased. Therefore, we have $C(S') \geq C(S'')$.

These steps are also described in Figure 5. Due to this transformation, we can assume that each job $i \in \hat{\tau}_t$ starts at time t_b.

The proof of Corollary 6.5 results from the combination of those four steps. ∎

To complete the proof of Theorem 6.2, we must address the processing times of the jobs in $\hat{\tau}_t$ and the machine requirements of all preemption-causing jobs.

We start by looking at the special case described in Figure 6. There, we transform the source job system into the target system in Table 4 with m' not necessarily being an integer. Further, we assume that $g > b > a \gg 1$ holds. This can always be achieved by scaling; see Corollary 6.4. Optimal schedules for both systems are given in Figure 6. In addition, we consider separate optimal schedules for the sequential jobs only and require that the machine-time product and the total weighted completion time remain invariant for both systems; that is, we have

$$(g+b-a)\frac{m}{2} = (d-c)m' \text{ and } (g^2+(b-a)^2)\frac{m}{2} = (d-c)^2 m'.$$

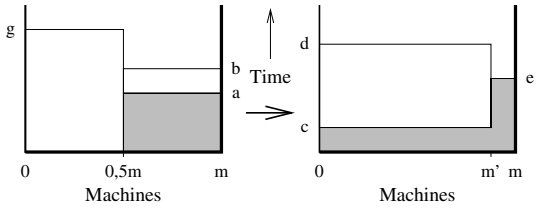


FIG. 6. *Source (left) and target (right) optimal schedules for the special case.*

The same must be valid for the parallel jobs in the schedules of Figure 6; that is,

$$a\frac{m}{2} = cm' + e(m-m') \text{ and } \frac{1}{2}a^2\frac{m}{2} = \frac{1}{2}(c^2m' + e^2(m-m')) \text{ hold.}$$

There are four equations to determine the four variables m', c, d, and e. Solving this system of equations results in

$$m' = \frac{m}{2}\frac{(g+b-a)^2}{g^2+(b-a)^2},\quad c = \frac{a(b-a)}{g+b-a},\quad d = c + \frac{g^2+(b-a)^2}{g+b-a},\quad e = \frac{ag}{g-b+a}.$$

There is no difference in the total weighted completion time of both schedules, as we have

$$a(b-a)\frac{m}{2} = c(d-c)m'.$$

Intuitively, we can therefore state that for $m \to \infty$, a job system containing m sequential jobs with two different processing times always can be transformed into a job system where all sequential jobs have the same processing time and the same total weighted completion times, and the total machine-time products of the optimal schedules remain invariant. This transformation can be applied repeatedly to handle job systems where several different processing times exist for the m sequential jobs. With this result, we are ready to prove the next corollary.

COROLLARY 6.6. *Assume a job system τ as described in Corollaries* 6.4 *and* 6.5 *with $\frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})} > 1 + \frac{1}{v} + p$ and a PSRS schedule S. Then there is another job system τ' and a PSRS schedule S' such that $\frac{\hat{C}(S')}{\tilde{C}^*(\hat{\tau}')} \geq \frac{\hat{C}(S)}{\tilde{C}^*(\hat{\tau})}$ and the following properties in addition to those of Corollaries* 6.4 *and* 6.5 *hold:*

(i) $p_i = p_j$ *for all jobs* $i, j \in \hat{\tau}'_t$;
(ii) $m_i = m - |\hat{\tau}'_t|$ *for all wide jobs* $i \in \hat{\tau}'$.

Finally, the job systems described by this corollary are the job systems of Table 3 if the additional k type 1 jobs are ignored in the PSRS schedule.

Proof. Using our lower bound approximations, the optimal schedule for job system $\hat{\tau}$ can be described by the top left schedule in Figure 7 where all jobs in $\hat{\tau}_t$ are shaded. Consider a machine with a larger makespan than the minimal makespan for all machines in this schedule. Such a machine executes a single job $j \in \hat{\tau}_t$ and possibly some parallel jobs from $\hat{\tau}$. We group all those machines such that the same number of parallel jobs is executed on each machine of a group and the sequential jobs on these machines have the same processing time.

We choose two groups and pick $\bar{m}$ machines from each group. $\bar{m}$ need not be an integer. The special case is applied to these $2\bar{m}$ machines. Note that it is not necessary

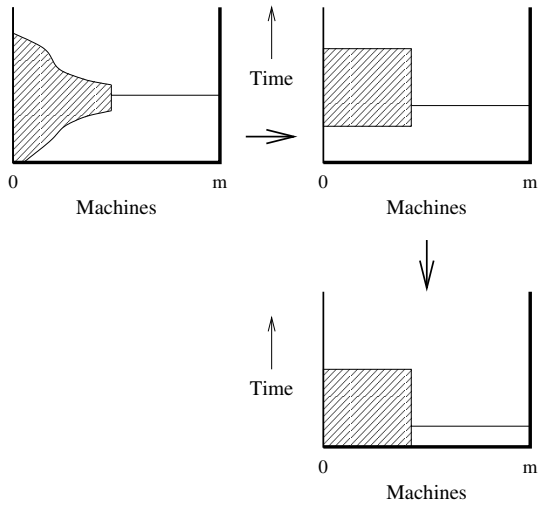


FIG. 7. *Transformation of* $\hat{\tau}'$.

to consider the parallel jobs that are executed on each of these $2\bar{m}$ machines. This procedure is applied repeatedly until we obtain the job system τ'' and the optimal schedule given by the top right schedule of Figure 7. The special case does not change the combined machine-time product of all parallel jobs together. However, as we have $e > a$ in the special case, it is necessary to combine some of the resulting jobs in order to generate parallel jobs, that is, to guarantee that in the optimal schedule, no parallel job is executed concurrently with any sequential job in $\hat{\tau}''\backslash\hat{\tau}''_t$. The processing time 1 for all parallel jobs can be achieved by using scaling and splitting; see the proof of Corollary 6.4.

Without taking into account any scaling, we have $\tilde{C}^*(\hat{\tau}) \geq \tilde{C}^*(\hat{\tau}'')$ as the number of machines not executing any job in $\hat{\tau}''_t$ is increased by the transformation, and their makespan is balanced.

The PSRS schedule S'' is obtained by ordering all parallel jobs $i, j \in \hat{\tau}''$ such that i precedes j in Q if $m_i > m_j$ holds, and arranging the sequential jobs in $\hat{\tau}''\backslash\hat{\tau}''_t$ accordingly to fit the conditions of Corollary 6.5. Therefore, S is transformed into S'' by repeatedly exchanging a sequential job $i \in \hat{\tau}\backslash\hat{\tau}_t$ of processing time $\frac{1}{v}$ with a part of a parallel job $j \in \hat{\tau}$ with processing time 1 and $t_i(3) < t_j(1)$ in S. As we have $v \leq 1$, this results in $\hat{C}(S) \leq \hat{C}(S'')$.

If there is any $j \in \hat{\tau}''$ with $m_j = m$, then we can assume that this job completes first in S''. Remember that this job will preempt a single sequential job i. However, this job i is ignored in our analysis. Next, we remove j. The repeated application of this procedure leads to job system $\hat{\tau}'$ and schedule S' as shown on the bottom of Figure 7. The removal of job j results in

$$\hat{C}(S'') - \hat{C}(S') = \left(1 + \frac{1}{v} + p\right) \sum_{i \in \hat{\tau}''} w_i$$

and

$$\tilde{C}^*(\hat{\tau}'') - \tilde{C}^*(\hat{\tau}') = \sum_{i \in \hat{\tau}''} w_i.$$

Finally, if there is any wide job i with $m_i < |\hat{\tau}_t|$, then we create a new job system τ' by enlarging i to i' with $m_{i'} = |\hat{\tau}_t|$ and by removing $|\hat{\tau}_t| - m_i$ of the sequential jobs preempted by i in S. As $v \leq 1$ holds, this procedure is similar to the removal of a sequential job; see step 3 in the proof of Corollary 6.5. Note that increasing the parallelism of any job up to $|\hat{\tau}_t|$ does not constrain the optimal schedule. □

The proof of Corollary 6.6 also concludes the proof of Theorem 6.2.

To obtain the minimal value $f_C(0) = 2.366$, we choose $v = 0.836$, $x = 0.183m$, and $y = 2$. Note that we have $f_C(0) > \frac{1}{v} + 1 = 2.196$.

7. Nonpreemptive scheduling. Preemptive schedules are sometimes used to construct nonpreemptive ones with good performance; see Phillips, Stein, and Wein [10] and Chekuri et al. [4]. As PSRS schedules have a good approximation factor compared to the best nonpreemptive methods, they are a good candidate for this approach. In addition, a PSRS schedule is a simple interleave of two nonpreemptive schedules which can easily be separated. Therefore, we address the transformation of a PSRS schedule into a nonpreemptive schedule in this section.

For the purpose of this transformation, we use the case $p = 0$. Further, $\tau_p(S)$ denotes the set of jobs that preempt at least one other job in a given PSRS schedule S. Remember that no job $i \in \tau \backslash \tau_p(S)$ is executed concurrently with any job in $\tau_p(S)$ in schedule S. Formally, we can therefore state that

$$t_i(2d_i + 1) \geq p_i + \sum_{j \in \tau_P(S) \wedge t_j(1) < t_i(2d_i+1)} p_j.$$

The transformation algorithm is given in Table 5. All schedule data used in this algorithm, like $t_i(2d_i + 1)$, refer to the PSRS schedule S. Intuitively, the algorithm generates two sequences of time frames, each of which covers the whole schedule. The time frames of each sequence increase in size with their position in the sequence. The second frame of the second sequence may be the only exception to this rule. The second sequence is related to all jobs in $\tau_p(S)$ while the first sequence covers all other jobs in τ. Finally, a nonpreemptive schedule is generated by interleaving both sequences and determining a new list of jobs. The order of this list is based on the time frame in which a job completes. For jobs completing in the same time frame, the order of the original priority list Q is used. Schedule S' depends on two parameters $\alpha > 1$ and $\Delta < 1$, which describe the rate of frame length increase and the offset between the time frames of different sequences, respectively. The algorithm further uses n as an iteration counter and the variables l_1, l_2, l_3, and l_4 to denote the bounds of the actual time frames. The transformation algorithm has the same time complexity as Algorithm PSRS.

For the construction of the nonpreemptive list schedule, it is important to remember that removing any job from the priority list in such a list schedule cannot increase the completion time of any other job in the schedule.

To evaluate the resulting nonpreemptive schedule S', we first determine the makespan of the schedule after iteration n.

COROLLARY 7.1. *After execution of iteration n of the transformation algorithm, the makespan of this part of the nonpreemptive schedule is upper bounded by*

$$\begin{aligned}\sum_{\mu=0}^{n} \sum_{\nu=0}^{\mu} \alpha^{\nu} + c_1 + c_2 - c_3 &= \sum_{\mu=0}^{n} \frac{\alpha^{\mu+1} - 1}{\alpha - 1} + c_1 + c_2 - c_3 \\ &= \frac{\alpha^{n+2} - \alpha - (n+1)(\alpha - 1)}{(\alpha - 1)^2} + c_1 + c_2 - c_3\end{aligned}$$

TABLE 5
Nonpreemptive transformation of a PSRS schedule.

```
Generate a PSRS schedule S for a job system τ and a priority list Q;
n = 0;
l_1 = 0;
l_3 = 0;
while (l_1 < max_{i∈τ} t_i(2d_i + 1)){
        l_2 = l_1 + α^n;
        list schedule all jobs i ∈ τ\τ_p(S) with l_1 < t_i(2d_i + 1) ≤ l_2 in the order given by Q;
        l_1 = l_2;
        l_4 = l_2 + Δα^{n+1};
        list schedule all jobs i ∈ τ_p(S) with l_3 < t_i(2d_i + 1) ≤ l_4 in the order given by Q;
        l_3 = l_4;
        n = n + 1;
}
```

with

(i) $c_1 = t_i(1) - l_2$ *for* $i \in \tau_p(S)$ *and* $t_i(0) < l_2 < t_i(1) \le l_4$,
(ii) $c_2 = \sum p_i$ *with* $i \in \tau_p(S)$ *and* $l_2 \le t_i(0) < t_i(1) \le l_4$, *and*
(iii) $c_3 = l_2 - t_i(0)$ *for* $i \in \tau_p(S)$ *and* $t_i(0) < l_2 < l_4 < t_i(1)$.

Note that for jobs from $\tau_p(S)$, the processing time, which falls into the time interval $[0, l_2]$, is considered in the double sum term. Therefore, the correction terms c_1 and c_2 are used for jobs from $\tau_p(S)$ that complete after l_2 but before l_4. Hence, they are scheduled in iteration n. The correction term c_3 addresses a job from $\tau_p(S)$ that starts before l_2 and finishes after l_4. Therefore, it is not yet scheduled but considered in the double sum term. Of course, if we have $c_3 \neq 0$, then $c_1 = c_2 = 0$ must hold.

Proof. The proof is done by induction over the iteration index n. The claim clearly holds for $n = 0$. Assume that it is also true for some n. In the next iteration, the first sequence will cover S up to time $l_2 = \sum_{\mu=0}^{n+1} \alpha^\mu = \frac{\alpha^{n+2}-1}{\alpha-1}$. Here l_1, l_2, l_3, and l_4 denote the bounds of the frames in this iteration $n + 1$.

For the makespan $\bar{C}_{\max}$ of a list schedule of all jobs j in $\tau_{n+1} = \{j \in \tau\backslash\tau_p(S) | l_1 < t_j(2d_j + 1) \le l_2\}$ with the order of Q, we have

$$\bar{C}_{\max} \le \max_{j \in \tau_{n+1}} \{t_j(2d_j + 1)\} - \sum_{i \in \tau_p(S) \wedge t_i(1) \le t_j(2d_j+1)} p_i$$

due to our observation above. Note that setting $\max_{j\in\tau_{n+1}}\{t_j(2d_j + 1)\} = l_2$ includes the processing time of all jobs from $\tau_p(S)$ that complete in the time interval $[0, l_2]$. Considering the jobs in $\tau_p(S)$ with $l_2 < t_i(1)$ and $t_i(0) < l_4$ completes the proof. ∎

Now, we give a bound on the completion time $t'_i(1)$ of job $i \in \tau\backslash\tau_p(S)$ in the nonpreemptive schedule S'. To this end, we assume that i is scheduled in iteration n; that is, $l_1 = \sum_{\mu=0}^{n-1} \alpha^\mu < t_i(2d_i + 1) \le \sum_{\mu=0}^{n} \alpha^\mu = l_2$ holds.

For $n = 0$, it is easy to see that we have $t'_i(1) \le t_i(2d_i + 1)$. In all other cases, there is

$$\begin{aligned}\frac{t'_i(1)}{t_i(2d_i+1)} &\le \frac{t_i(2d_i+1) + \sum_{\mu=0}^{n-1} \frac{\alpha^{\mu+1}-1}{\alpha-1} + c_1 + c_2 - c_3}{t_i(2d_i+1)} \\ &\le 1 + \frac{\sum_{\mu=0}^{n-1} \frac{\alpha^{\mu+1}-1}{\alpha-1} + \Delta\alpha^n}{\sum_{\mu=0}^{n-1} \alpha^\mu}\end{aligned}$$

$$= 1 + \frac{\alpha}{\alpha - 1} + \frac{\Delta(\alpha - 1)\alpha^n - n}{\alpha^n - 1}$$
$$\leq 1 + \frac{\alpha}{\alpha - 1} + \Delta(\alpha - 1) \quad \text{for } \Delta(\alpha - 1) \leq n.$$

Similarly, a bound on the completion time $t'_i(1)$ in S' is calculated for any job $i \in \tau_p(S)$. Again, it is assumed that i is scheduled in iteration n with $l_3 = \sum_{\mu=0}^{n-1} \alpha^\mu + \Delta\alpha^n < t_i(1) \leq \sum_{\mu=0}^{n} \alpha^\mu + \Delta\alpha^{n+1} = l_4$. Remember that for all jobs from $\tau \backslash \tau_p(S)$, the processing time, which falls into the time interval $[0, l_2]$ in schedule S, is already included in the term $\sum_{\mu=0}^{n} \frac{\alpha^{\mu+1}-1}{\alpha-1}$.

For $n = 0$, there is $\frac{t'_i(1)}{t_i(1)} \leq \frac{1+\Delta\alpha}{1+\frac{1}{v}}$. In the general case, we have

$$\begin{aligned}
\frac{t'_i(1)}{t_i(1)} &\leq \frac{\max\{0, t_i(1) - l_2\} + \sum_{\mu=0}^{n} \frac{\alpha^{\mu+1}-1}{\alpha-1}}{t_i(1)} \\
&\leq \frac{\sum_{\mu=0}^{n} \frac{\alpha^{\mu+1}-1}{\alpha-1}}{\sum_{\mu=0}^{n-1} \alpha^\mu + \Delta\alpha^n} \\
&= \frac{\alpha^{n+2} - 1 - (\alpha - 1)(n + 2)}{(\alpha - 1)(\alpha^n - 1 + \Delta\alpha^n(\alpha - 1))} \\
&\leq \frac{\alpha^2}{(\alpha - 1)(1 + \Delta(\alpha - 1))} \quad \text{for } \alpha + 1 - \Delta \leq (1 + \Delta(\alpha - 1))(n + 2).
\end{aligned}$$

Therefore, we can derive the following performance guarantee for the nonpreemptive schedule.

THEOREM 7.2. *If* $\Delta(\alpha - 1) \leq 1$, $\alpha + 1 - \Delta \leq 3(1 + \Delta(\alpha - 1))$, *and* $\frac{1+\Delta\alpha}{1+\frac{1}{v}} \leq \frac{\alpha^2}{(\alpha-1)(1+\Delta(\alpha-1))}$ *hold, then the transformation of a PSRS schedule* S *into a nonpreemptive schedule* S' *guarantees*

$$\frac{C(S')}{C^*} \leq \max\left\{1 + \frac{\alpha}{\alpha - 1} + \Delta(\alpha - 1), \frac{\alpha^2}{(\alpha - 1)(1 + \Delta(\alpha - 1))}\right\} f_c(0).$$

Proof. The proof is a direct consequence of the statements made above. ∎

For $\Delta = 0.25$ and $\alpha = 3$, we obtain $\frac{C(S')}{C^*} \leq 3 \times 2.37 = 7.11$. The validity of the additional conditions of Theorem 7.2 can easily be checked. Similarly, a makespan performance guarantee can be determined from the proof of Theorem 7.2.

8. Conclusion. First, we addressed bicriteria scheduling of parallel jobs in general and gave a few new results. Then we presented an algorithm that generates preemptive offline schedules for parallel and independent jobs with fixed resource requirements. This algorithm is obtained by combining two algorithms with good performance for restricted input sets. The schedule is based on a priority list and has small approximation factors for both total weighted completion time and makespan criteria. It does not require job migration. The method belongs to the class of list scheduling algorithms. It is carefully analyzed and a tight worst-case approximation factor is determined. Moreover, the analysis provides information about the structure of "bad cases." Also, we derived a numerical optimization problem which can be used to fine-tune the total weighted completion time approximation factor. Finally, we transformed the preemptive schedules into nonpreemptive ones with a better bound for total weighted completion time than those obtained by previously known methods.

TABLE 6
Comparison between various scheduling methods.

Schedule	Type	$\frac{C(S)}{C^*}$	$\frac{C_{\max}(S)}{C^*_{\max}}$
SMART	nonpreemptive	8.53	5.19
SMART	nonpreemptive	9	5
PSRS	nonpreemptive	7.11	9.60
PSRS	nonpreemptive	7.26	9.00
PSRS with $p = 0$	preemptive	2.37	3.20
PSRS with $p = 0$	preemptive	2.42	3
PSRS with $p = 1$	preemptive	3.41	4.31
PSRS with $p = 1$	preemptive	3.61	4

The generated schedules are based upon our machine model which is derived from existing parallel computers. To our knowledge, it is also the first time that a preemption penalty is considered in the analysis of such an algorithm. Compared with nonpreemptive SMART schedules, our approximation factors are significantly better, even if we assume that a context switch is as time consuming as the minimal completion time of a job including loading the job and storing its results, that is, if $p = 1$ holds. As shown in Table 6, PSRS and SMART schedules can be fine-tuned to minimize either $\frac{C(S)}{C^*}$ or $\frac{C_{\max}(S)}{C^*_{\max}}$.

PSRS schedules have the additional advantage that they use preemption only for jobs which require at most 50% of the nodes. Even in this case, there are at most two jobs resident on any node at the same time. Moreover, PSRS schedules only need global preemption which may be easier to implement than other forms of gang scheduling with respect to running messages in the interconnection network.

Acknowledgments. The author is grateful to Joel Wein for a helpful discussion on bicriteria scheduling. The author would also like to thank the anonymous referees for their helpful remarks.

REFERENCES

[1] F. AFRATI, E. BAMPIS, C. CHEKURI, D. KARGER, C. KENYON, S. KHANNA, I. MILIS, M. QUEYRANNE, M. SKUTELLA, C. STEIN, AND M. SVIRIDENKO, *Approximation schemes for minimizing average weighted completion time with release dates*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, 1999, pp. 32–43.

[2] J. BRUNO, E. COFFMAN, AND R. SETHI, *Scheduling independent tasks to reduce mean finishing time*, Commun. ACM, 17 (1974), pp. 382–387.

[3] S. CHAKRABARTI, C. PHILLIPS, A. SCHULZ, D. SHMOYS, C. STEIN, AND J. WEIN, *Improved approximation algorithms for minsum criteria*, in Proceedings of the 1996 International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 1099, F. M. auf der Heide and B. Monien, eds., Springer-Verlag, New York, 1996, pp. 646–657.

[4] C. CHEKURI, R. MOTWANI, B. NATARAJAN, AND C. STEIN, *Approximation techniques for average completion time scheduling*, SIAM J. Comput., 31 (2001), pp. 146–166.

[5] X. DENG, N. GU, T. BRECHT, AND K. LU, *Preemptive scheduling of parallel jobs on multiprocessors*, SIAM J. Comput., 30 (2000), pp. 145–160.

[6] D. FEITELSON AND L. RUDOLPH, *Parallel job scheduling: Issues and approaches*, in IPPS' 95 Workshop: Job Scheduling Strategies for Parallel Processing, Lecture Notes in Comput. Sci. 949, D. Feitelson and L. Rudolph, eds., Springer-Verlag, New York, 1995, pp. 1–18.

[7] A. Fishkin, K. Jansen, and L. Porkolab, *On minimizing average weighted completion time of multiprocessor tasks with release dates*, in Proceedings of the 28th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2076, F. Orejas, P. Spirakis, and J. van Leeuwen, eds., Springer-Verlag, New York, 2001, pp. 875–886.

[8] T. Kawaguchi and S. Kyan, *Worst case bound of an LRF schedule for the mean weighted flow-time problem*, SIAM J. Comput., 15 (1986), pp. 1119–1129.

[9] R. McNaughton, *Scheduling with deadlines and loss functions*, Management Sci., 6 (1959), pp. 1–12.

[10] C. Phillips, C. Stein, and J. Wein, *Minimizing average completion time in the presence of release dates*, Math. Programming, 82 (1998), pp. 199–223.

[11] U. Schwiegelshohn, W. Ludwig, J. Wolf, J. Turek, and P. Yu, *Smart SMART bounds for weighted response time scheduling*, SIAM J. Comput., 28 (1998), pp. 237–253.

[12] W. Smith, *Various optimizers for single-stage production*, Naval Res. Logist., 3 (1956), pp. 59–66.

[13] C. Stein and J. Wein, *On the existence of schedules that are near-optimal for both makespan and total weighted completion time*, Oper. Res. Lett., 21 (1997), pp. 115–122.

[14] J. Turek, W. Ludwig, J. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. Yu, *Scheduling parallelizable tasks to minimize average response time*, in Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architectures, Cape May, NJ, 1994, pp. 200–209.

[15] J. Turek, U. Schwiegelshohn, J. Wolf, and P. Yu, *Scheduling parallel tasks to minimize average response time*, in Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 112–121.

SIAM J. COMPUT.
Vol. 33, No. 6, pp. 1309–1337

ALGEBRAIC PROPERTIES FOR SELECTOR FUNCTIONS*

LANE A. HEMASPAANDRA†, HARALD HEMPEL‡, AND ARFST NICKELSEN§

Abstract. The *nondeterministic* advice complexity of the P-selective sets is known to be exactly linear. Regarding the *deterministic* advice complexity of the P-selective sets—i.e., the amount of Karp–Lipton advice needed for polynomial-time machines to recognize them in general—the best current upper bound is quadratic [K. Ko, *J. Comput. System Sci.*, 26 (1983), pp. 209–221] and the best current lower bound is linear [L. Hemaspaandra and L. Torenvliet, *Theoret. Comput. Sci.*, 154 (1996), pp. 367–377].

We prove that every associatively P-selective set is commutatively, associatively P-selective. Using this, we establish an algebraic sufficient condition for the P-selective sets to have a linear upper bound (which thus would match the existing lower bound) on their deterministic advice complexity: If all P-selective sets are associatively P-selective, then the deterministic advice complexity of the P-selective sets is linear. The weakest previously known sufficient condition was P = NP.

We also establish related results for algebraic properties of, and advice complexity of, the nondeterministically selective sets.

Key words. computational complexity theory, P-selectivity, NP-selectivity, nondeterministic selectivity, selector functions, advice complexity, nonuniform complexity, semifeasible computation, algebraic properties, associativity, commutativity, immunity, printability, tournaments, digraphs

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

DOI. 10.1137/S0097539703427550

1. Introduction.

Selman [Sel79, Sel81, Sel82a, Sel82b] defined the P-selective sets about twenty years ago. In addition to being of interest in their own right, they have recently had some surprising applications. For example, selectivity is a powerful tool in the study of search versus decision problems [HNOS96a], and nondeterministic generalizations of selectivity are the key tools used to show that even NP machines cannot uniquely refine satisfying assignments unless the polynomial hierarchy collapses [HNOS96b], that even weaker refinements are also precluded unless the polynomial hierarchy collapses [Ogi96, NRRS98], and that many cardinality types of nondeterministic function classes cannot collapse unless the polynomial hierarchy collapses [HOW02].

DEFINITION 1.1 (see [Sel79]). *A set B is* P*-selective if and only if there is a (total) polynomial-time function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that*

1. $(\forall x, y)[f(x, y) = x \ \vee \ f(x, y) = y]$, *and*
2. $(\forall x, y)[(x \in B \ \vee \ y \in B) \implies f(x, y) \in B]$.

We call such a function f a P*-selector function for B.*

*Received by the editors May 8, 2003; accepted for publication (in revised form) January 20, 2004; published electronically August 6, 2004. Earlier versions of some parts of this paper were presented at the COCOON 2001 conference.

http://www.siam.org/journals/sicomp/33-6/42755.html

†Department of Computer Science, University of Rochester, Rochester, NY 14627-0226 (lane@cs.rochester.edu). This author's work was supported in part by grants NSF-CCR-9322513 and NSF-INT-9815095/DAAD-315-PPP-gü-ab.

‡Institut für Informatik, Friedrich-Schiller-Universität Jena, D-07743 Jena, Germany (hempel@informatik.uni-jena.de). This author's work was supported in part by grant NSF-INT-9815095/DAAD-315-PPP-gü-ab and was done while the author was visiting the University of Rochester under a NATO Postdoctoral Science Fellowship from the Deutscher Akademischer Austauschdienst's "Gemeinsames Hochschulsonderprogramm III von Bund und Ländern" program.

§Fachbereich Informatik, Technische Universität Berlin, D-10587 Berlin, Germany (nicke@cs.tu-berlin.de).

That is, a set B is P-selective if there is a polynomial-time function f that, given any two strings, always chooses one of them, and f does this in such a way that if exactly one of the strings is in B, then the P-selector function chooses that string. $f(x, y)$ is often described, very informally, as choosing one of x or y that is more likely to be in the set, though a more accurate description would be that f chooses one of x or y that is logically no less likely than the other to be in the set.

The P-selective sets have been extensively studied, and much about them is well understood (see the recent book [HT03] about selectivity theory). Though some P-selective sets are very complex—highly undecidable—the P-selective sets nonetheless have a broad range of structural simplicity properties. Most crucially in terms of the study in this paper, Ko [Ko83] showed that they have low nonuniform complexity (P-sel $\subseteq$ P/$\mathcal{O}(n^2)$). The following are a few of the many other simplicity results known to hold: Ko and Schöning [KS85] showed that all P-selective sets in NP are in the second level of the low hierarchy of Schöning [Sch83], and Allender and Hemachandra [AH92] showed that the Ko–Schöning result is the strongest lowness result for P-sel that holds with respect to all oracles; a long line of work starting with Selman [Sel79] and Toda [Tod91] (see also [Siv99] and the references therein) has shown that no P-selective set can be NP-hard under $\leq^{\mathrm{p}}_m$ or various other reductions unless P = NP; Naik and Selman [NS99] have shown that no P-selective set can be truth-table-hard for NP unless certain (intuitively unlikely) containments hold in the relationship between adaptive and nonadaptive queries to NP; and as a consequence of the work of Ko [Ko83] and Cai [Cai01] no P-selective set can be truth-table-hard for NP (or even Turing-hard for NP) unless the polynomial hierarchy collapses to S_2, where S_2 is the symmetric alternation class of Canetti [Can96] and Russell and Sundaram [RS98]. Note that $\mathrm{S}_2 \subseteq \mathrm{ZPP}^{\mathrm{NP}} \subseteq \mathrm{NP}^{\mathrm{NP}}$ [Cai01], where ZPP as usual denotes expected polynomial time, so this is a very dramatic collapse.

In this paper, we show that sets having P- or NP-selector functions with nice algebraic properties have simplicity properties far beyond those known for general P-selective or NP-selective sets. More generally, we study the class of languages one obtains from P- and NP-selector functions having certain algebraic properties, and the possibility of obtaining such algebraically nice P- or NP-selector functions.

In particular, section 4 shows that any P-selective (NP-selective) set having an associative P-selector (NP-selector) function also has a commutative, associative P-selector (NP-selector) function. Section 5 shows that sets having an associative P-selector can be accepted by deterministic advice interpreters using only a linear amount of advice. In contrast, the best upper bound on the deterministic advice complexity of the P-selective sets is the quadratic bound obtained by Ko [Ko83].

Our result provides a new sufficient condition—all P-selective sets are associatively P-selective—for all P-selective sets having linear deterministic advice; the weakest previously known sufficient condition was the quite demanding assumption that P = NP.

Section 6 shows that associatively NP-selective sets with weak census functions cannot be coNP-immune. Section 7 establishes a structural sufficient condition for all P-selective sets being associatively, commutatively P-selective, and proves that if all NPMV-selective sets have associative NPMV-selector functions, then the polynomial hierarchy collapses.

2. Definitions. Commutativity and associativity are concepts often associated with single-valued total functions. However, as we will discuss soon, these concepts can be naturally applied to multivalued and/or partial functions (see, e.g., [RS97, HR99]).

For a 2-ary multivalued function f, let *set-*$f(x, y)$ denote the set of values of f on input (x, y).[1] $f(x, y)$ is undefined if and only if *set-*$f(x, y) = \emptyset$. We extend this notation to single-valued functions in the obvious way. For a 2-ary single-valued function f let *set-*$f(x, y) = \{f(x, y)\}$ if $f(x, y)$ is defined and let *set-*$f(x, y) = \emptyset$ otherwise. For a (single- or multivalued) function f, a set A, and a string y, define *set-*$f(A, y) = \bigcup_{a \in A}$ *set-*$f(a, y)$ and *set-*$f(y, A) = \bigcup_{a \in A}$ *set-*$f(y, a)$. A 2-ary function f is called *total on a set* B if and only if $f(x, y)$ is defined for all $x, y \in B$. A function is called total if it is total on Σ^*.

A total 2-ary (single- or multivalued) function f is *associative on a set* B if and only if *set-*$f(a,$ *set-*$f(b, c)) =$ *set-*$f($*set-*$f(a, b), c)$ holds for all $a, b, c \in B$.[2] (Note that for total single-valued functions this is equivalent to saying that $f(a, f(b, c)) = f(f(a, b), c)$ holds for all $a, b, c \in B$.) We say a total function f is *associative* if f is associative on Σ^*. A total 2-ary (single- or multivalued) function f is *associative at each length* if f is associative on Σ^n for each n (where Σ^n is the set of words of length n). A 2-ary function f (partial or total, single- or multivalued) is *commutative* if and only if *set-*$f(a, b) =$ *set-*$f(b, a)$ holds for all a and b. (For total single-valued functions this is equivalent to saying that $f(a, b) = f(b, a)$ holds for all a, b.)

For partial functions the literature distinguishes two notions of associativity, namely, (strong) associativity and weak associativity [HR99, RS97]. A partial 2-ary function f is called associative (or strongly associative) if and only if *set-*$f(a,$ *set-*$f(b, c))$ $=$ *set-*$f($*set-*$f(a, b), c)$ holds for all a, b, and c. A partial 2-ary function f is called weakly associative if and only if for all a, b, and c, either (a) *set-*$f(a,$ *set-*$f(b, c)) = \emptyset$ or *set-*$f($*set-*$f(a, b), c) = \emptyset$, or (b) *set-*$f(a,$ *set-*$f(b, c)) =$ *set-*$f($*set-*$f(a, b), c)$. In other words, if any of the four applications of f are undefined, weak associativity "forgives" the equality requirement. The two approaches to associativity for functions correspond to the two notions of equality for partial functions, which date back to the work of Kleene [Kle52]. Note that both of these definitions, for total functions, exactly coincide with the associativity definition for total functions given earlier.

DEFINITION 2.1 (see [HHN+95, HNOS96b]). *For any class of (partial or total, single- or multivalued) functions $\mathcal{F}$, we say a set B is $\mathcal{F}$-selective exactly if there is a function $f \in \mathcal{F}$ such that*

1. $(\forall x, y)[$*set-*$f(x, y) \subseteq \{x, y\}]$, *and*
2. $(\forall x, y)[\{x, y\} \cap B \neq \emptyset \implies ($*set-*$f(x, y) \neq \emptyset \wedge$ *set-*$f(x, y) \subseteq B)]$.

A function f with property 1 *is called a self-contained function. A function with properties* 1 *and* 2 *is called an $\mathcal{F}$-selector function for B (when B is clear from context or unimportant, we will simply speak of an $\mathcal{F}$-selector function, and when $\mathcal{F}$ is clear from context or unimportant, we will simply speak of a selector function (for B)).*

[1] Formally, one should speak of relations rather than functions. However, the longstanding convention in computer science is to refer to such objects as "multivalued functions" (see [BLS84, BLS85, Sel96]). Similarly, the notation "*set-*$f(...)$" is also standard in the literature on multivalued functions, and we follow it. Though this notation makes equations a bit longer and less beautiful (than in the reasonable alternative, which some people prefer, of viewing f as standing for *set-*f), it does avoid some confusion. For example, suppose f is undefined when its first argument is x_0 and its second argument is y_0. Then we would say that $f(x_0, y_0)$ is undefined. In contrast, *set-*$f(x_0, y_0)$ certainly *is* defined, and in particular *set-*$f(x_0, y_0) = \emptyset$. Readers particularly interested in the history, types, and notation involving multivalued functions may wish to look at the interesting papers of Book, Long, and Selman mentioned earlier in this footnote.

[2] As it stands, the definition allows the possibility that some of the values or all of the values in *set-*$f(b, c)$, *set-*$f(a, b)$, *set-*$f(a,$ *set-*$f(b, c))$, and *set-*$f($*set-*$f(a, b), c)$ might not belong to B. However, since we will typically be applying this to total, *self-contained* functions, the four just-mentioned sets will each be nonempty subsets of B.

This definition requires a selector function for a set B to be defined whenever at least one input is in B. In other words, partial selector functions can be undefined only if both inputs are in $\overline{B}$. Observe that for any *total* self-contained function f we have that for any strings a, b, and c,

$$\textit{set-}f(a,b) \cap \textit{set-}f(a,c) \subseteq \textit{set-}f(a,\textit{set-}f(b,c)) \subseteq \textit{set-}f(a,b) \cup \textit{set-}f(a,c)$$

and

$$\textit{set-}f(a,c) \cap \textit{set-}f(b,c) \subseteq \textit{set-}f(\textit{set-}f(a,b),c) \subseteq \textit{set-}f(a,c) \cup \textit{set-}f(b,c).$$

We will use the following notational shorthand for the classes that we will study. (The ordering of "A" and "C" changes to avoid confusion with the existing class "AC," and to make clear that the "ℓ" modifies just the "A.")

DEFINITION 2.2. *For any class of (partial or total, single- or multivalued) functions $\mathcal{F}$,*

1. $\mathcal{F}\textit{-sel} = \{B \mid B \textit{ is } \mathcal{F}\textit{-selective}\}$,
2. $A\textit{-}\mathcal{F}\textit{-sel} = \{B \mid B \textit{ is } \mathcal{F}\textit{-selective via an associative } \mathcal{F}\textit{-selector function}\}$,
3. $C\textit{-}\mathcal{F}\textit{-sel} = \{B \mid B \textit{ is } \mathcal{F}\textit{-selective via a commutative } \mathcal{F}\textit{-selector function}\}$,
4. $CA\textit{-}\mathcal{F}\textit{-sel} = \{B \mid B \textit{ is } \mathcal{F}\textit{-selective via a commutative, associative } \mathcal{F}\textit{-selector function}\}$,
5. $A_\ell\textit{-}\mathcal{F}\textit{-sel} = \{B \mid B \textit{ is } \mathcal{F}\textit{-selective via an } \mathcal{F}\textit{-selector function that is associative at each length}\}$,
6. $A_\ell C\textit{-}\mathcal{F}\textit{-sel} = \{B \mid B \textit{ is } \mathcal{F}\textit{-selective via a commutative } \mathcal{F}\textit{-selector function that is associative at each length}\}$.

FP will denote the (possibly partial) polynomial-time computable functions. $\mathrm{FP_t}$ will denote those functions in FP that are total. (In the literature, some authors use the notation "FP" to denote what we in this paper denote $\mathrm{FP_t}$. To avoid confusion, we urge the reader to note carefully that throughout this paper FP (and also the function classes NPSV and NPMV to soon be defined) do not require totality except when subscripted with a "t", i.e., $\mathrm{FP_t}$, $\mathrm{NPSV_t}$, and $\mathrm{NPMV_t}$.)

For any set C, FP^C will be the class of all (possibly partial) functions that are computable in polynomial time with the help of oracle C. For any class $\mathcal{C}$, $\mathrm{FP}^{\mathcal{C}} = \{g \mid (\exists C \in \mathcal{C})[g \in \mathrm{FP}^C]\}$. ${\mathrm{FP_t}}^C$ and ${\mathrm{FP_t}}^{\mathcal{C}}$ are defined analogously.

FP and $\mathrm{FP_t}$ functions each have some fixed arity for their domain, and their codomain is of arity one and is Σ^*. That is, FP and $\mathrm{FP_t}$ are, respectively, potentially-partial and per-force-total functions, each of type $\Sigma^* \to \Sigma^*$, or of type $\Sigma^* \times \Sigma^* \to \Sigma^*$, or of type $\Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^*$, etc. However, for selector functions, which are the main focus of this paper, the (input) arity is always two—selector functions by their nature are two-argument functions. So, throughout this paper, whenever we speak of selector-type functions (or related functions that by their nature or the way we use them are clearly 2-ary), we will simply write expressions such as "there is an $f \in \mathrm{FP}$" when in fact we mean "there is a 2-ary function $f \in \mathrm{FP}$."

We will follow the same convention regarding the nondeterministic function classes NPSV, $\mathrm{NPSV_t}$, NPMV, and $\mathrm{NPMV_t}$. These classes in general are meaningful for 1-ary functions, for 2-ary functions, etc. However, since this paper focuses on selector functions, which are always 2-ary, we will in effect silently treat the classes as if they are over 2-ary functions. So, for example, in the following paragraph, our machines have two arguments. (The case of k-ary, $k \neq 2$, functions belonging to these classes is identically defined except with k arguments.) And throughout this paper we simply

say, for our nondeterministic function classes, things such as "there is an $f \in \mathrm{NPSV}$" when in fact we mean "there is a 2-ary function $f \in \mathrm{NPSV}$."

In the literature (and in the present paper), $\mathrm{FP_t}$-sel is often denoted P-sel (note that Definition 2.2 for $\mathcal{F} = \mathrm{FP_t}$ yields the same class of P-selective sets as defined in Definition 1.1), and $\mathrm{NPSV_t}$-sel is often denoted NP-sel. (Both $\mathrm{FP_t}$-sel and $\mathrm{NPSV_t}$-sel in fact, by the convention mentioned in the previous paragraph, actually are intended to refer to functions of type $\Sigma^* \times \Sigma^* \to \Sigma^*$ from $\mathrm{FP_t}$ and $\mathrm{NPSV_t}$, respectively.)

We now define the standard nondeterministic function classes [BLS84, BLS85]. For any nondeterministic polynomial-time Turing machine (NPTM, for short) M taking two strings as its input and for any two strings x and y, let $out_M(x, y)$ denote the set of all strings that are output on some accepting path of $M(x, y)$, by which we mean M with x as its first argument and y as its second argument. Note that we view the collection of outputs as a set, not as a multiset (if one path outputs 101 and seven paths output 1100, the set of outputs is simply $\{101, 1100\}$), and note also that the functions computed by such machines may in some cases be partial functions, and may in some cases be multivalued functions. NPSV is the class of all single-valued functions f such that there exists an NPTM M such that, for all $x, y \in \Sigma^*$, $out_M(x, y) = \{f(x, y)\}$ if $f(x, y)$ is defined and $out_M(x, y) = \emptyset$ if $f(x, y)$ is undefined. NPMV is the class of all functions f such that there exists an NPTM M such that, for all $x, y \in \Sigma^*$, $out_M(x, y) = set\text{-}f(x, y)$. $\mathrm{NPSV_t}$ and $\mathrm{NPMV_t}$ denote the set of all total NPSV and NPMV functions, respectively.

As to levels of the polynomial hierarchy [MS72, Sto76] beyond NP, as is standard we will use Σ_2^p to denote the class of all languages that can be accepted by nondeterministic polynomial-time Turing machines with the help of an NP oracle, i.e., $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$. And as is standard we will use Σ_3^p to denote the class of all languages that can be accepted by nondeterministic polynomial-time Turing machines with the help of a Σ_2^p oracle, i.e., $\Sigma_3^p = \mathrm{NP}^{\Sigma_2^p} = \mathrm{NP}^{\mathrm{NP}^{\mathrm{NP}}}$.

Just as P-sel and NP-sel are standard notational conventions when $\mathrm{FP_t}$-sel and $\mathrm{NPSV_t}$-sel are actually meant (see [Sel79] and [HHN+95]), we similarly write A-P-sel and A-NP-sel as shorthand for A-$\mathrm{FP_t}$-sel and A-$\mathrm{NPSV_t}$-sel, respectively, and we do likewise for the other four prefixes C-, CA-, A_ℓ-, and A_ℓC-.

From the definitions,

$$\text{CA-P-sel} \subseteq \left\{\begin{array}{c}\subseteq \text{A-P-sel} \subseteq \\ \subseteq \mathrm{A}_\ell\text{C-P-sel} \subseteq\end{array}\right\} \mathrm{A}_\ell\text{-P-sel} \subseteq \text{P-sel}$$

and

$$\mathrm{A}_\ell\text{C-P-sel} \subseteq \text{C-P-sel} \subseteq \text{P-sel}.$$

Exactly the same inclusion relations hold for the various subtypes of NP-selective and of $\mathrm{NPMV_t}$-selective sets.

The following facts are well known.

FACT 2.3.

1. FP-sel = P-sel = C-P-sel.
2. NP-sel = C-NP-sel.
3. NPSV-sel = C-NPSV-sel.
4. $\mathrm{NPMV_t}$-sel = C-$\mathrm{NPMV_t}$-sel.
5. NPMV-sel = C-NPMV-sel.

Regarding the first equality of part 1 of Fact 2.3, recall that any partial selector function can only be undefined if both of its inputs are not in the selected set. Of course one can alter a deterministic polynomial-time Turing machine computing a partial selector function f for a set A in such a way that it outputs any one of the two inputs, say the lexicographically larger one, in the case that it "plans" to output nothing. The total function computed by the altered Turing machine is also a selector function for A. The other facts hold since if a set B is $\mathcal{F}$-selective via an $\mathcal{F}$-selector function f (with $\mathcal{F} \in \{\mathrm{FP_t}, \mathrm{NPSV_t}, \mathrm{NPSV}, \mathrm{NPMV_t}, \mathrm{NPMV}\}$), then the function f' defined by

$$(*) \qquad f'(x,y) = f(\min(x,y), \max(x,y))$$

is a commutative $\mathcal{F}$-selector function for B.

Advice classes capture the information content of sets with respect to some complexity class of decoder sets.

DEFINITION 2.4 (see [KL80]).

1. *For a complexity class $\mathcal{C}$ and a function $f : \mathbb{N} \to \mathbb{N}$, a set B is in $\mathcal{C}/f(n)$ if and only if there exist a set $C \in \mathcal{C}$ and a function $h : 1^* \to \Sigma^*$ such that, for all $x \in \Sigma^*$,*
   (a) $|h(1^{|x|})| = f(|x|)$, *and*
   (b) $x \in B \iff \langle x, h(1^{|x|})\rangle \in C$.
2. *For any complexity class $\mathcal{C}$ and any function class $\mathcal{F}$, $\mathcal{C}/\mathcal{F} = \bigcup_{f\in\mathcal{F}} \mathcal{C}/f(n)$.*

In this paper, we are particularly interested in the advice classes $\mathrm{P}/\mathcal{O}(n)$, $\mathrm{P}/n{+}1$, P/n, $(\mathrm{NP}\cap\mathrm{coNP})/n{+}1$, and $\mathrm{NP}/n{+}1$. Also, poly denotes the class of polynomials mapping from $\mathbb{N}$ to $\mathbb{N}$, and we will use later classes of the form $\mathcal{C}/\mathrm{poly}$.

For each set of strings A and each natural number n, let $A^{=n}$ and $A^{\leq n}$ denote all strings in A of length exactly n and of length up to and including n, respectively. Let $\mathcal{F}$ be any function class. A set B is *$\mathcal{F}$-printable* if there is a function $g \in \mathcal{F}$ such that, for all $n \in \mathbb{N}$, $g(1^n)$ outputs $B^{\leq n}$ in some fixed standard way of encoding sets as strings [HY84]. That is, there is a function from $\mathcal{F}$ that finds all elements in the set up to a given length. By tradition, P-printable denotes the $\mathrm{FP_t}$-printable sets.

Consider a relation R over the universe U, i.e., $R \subseteq U \times U$. We say that R is reflexive if and only if $(\forall a \in U)[(a,a) \in R]$. R is said to be antisymmetric if and only if $(\forall a, b \in U)[((a,b) \in R \wedge (b,a) \in R) \implies a = b]$. We say that R is a *transitive relation* if and only if $(\forall a, b, c \in U)[((a,b) \in R \wedge (b,c) \in R) \implies (a,c) \in R]$. For each $a, b \in U$, a and b are called comparable with respect to R if $(a,b) \in R \vee (b,a) \in R$. R is called a partial order on U if and only if R is reflexive, antisymmetric, and transitive. (Informally, order mirrors the relation "less than or equal to.") R is called a linear order (or total order, or, for short, we may sometimes simply write order) on U if and only if R is a partial order on U and each pair of elements from U is comparable with respect to R. Given a set U and a relation $\leq$ that is a partial order on U, (a) an element $x \in U$ is said to be maximal if and only if $(\forall y \in U)[x \leq y \implies x = y]$; (b) an element $x \in U$ is said to be minimal if and only if $(\forall y \in U)[y \leq x \implies x = y]$. If a maximal (minimal) element exists in a linear order it is unique and can be referred to as maximum (minimum). (For a review of the basics of orders see, e.g., [Cam94, Chapter 12].)

As is standard in graph theory, in a directed graph (digraph for short) $G = (V, E)$, we have that V is a finite set and $E \subseteq V \times V$. A digraph $G = (V, E)$ is called a *tournament* if and only if (a) for all $u, v \in V$ such that $u \neq v$, exactly one of the edges (u, v) and (v, u) belongs to E, and (b) for all $u \in V$, it holds that $(u, u) \notin E$. We will

in this paper slightly modify this notion by adding self-loops at each node, that is, a digraph $G = (V, E)$ is called a *self-looped tournament* (s-tournament, for short) if and only if (a) for all $u, v \in V$ such that $u \neq v$, exactly one of the edges (u, v) and (v, u) belongs to E, and (b) for all $u \in V$, it holds that $(u, u) \in E$. (If we were dealing only with tournament-type graphs, this modification would not be useful. However, in this paper we study even multivalued selector functions, and the natural graphs that these induce are of a different flavor, and in particular when they are transitive and contain cycles will have self-loops along all cycles even if we do not build self-loops into the way we induce these graphs. Thus, to allow the similarity of the relationship—which we will later see—between associativity of selector functions and transitivity of selector-induced digraphs to be as clear as possible, we use s-tournaments rather than tournaments.)

A digraph $G = (V, E)$ is called a *complete digraph* (or synonymously a *weak clique*) if and only if, for all $u, v \in V$, *at least* one of the edges (u, v) and (v, u) belongs to E. A digraph $G = (V, E)$ is called a *strong clique* if and only if, for all $u, v \in V$, the edge (u, v) belongs to E. (Note that both complete digraphs and strong cliques by these definitions must have every self-loop.) A subgraph G' of G is called a maximal strong clique if G' is a strong clique and there is no subgraph G'' in G that is a strong clique and contains G' as a proper subgraph.

As is standard, given a digraph $G = (V, E)$, a digraph $G' = (V', E')$ is said to be an *induced subgraph of* G exactly if $V' \subseteq V$ and $E' = \{(a, b) \mid a \in V' \wedge b \in V' \wedge (a, b) \in E\}$.

3. Commutative selectors and digraphs. Parts 1, 2, and 4 of Fact 2.3 establish that without loss of generality one can assume P- (NP- or $\mathrm{NPMV_t}$-) selectors to be commutative. Or, to be more precise, they make clear that restricting oneself to commutative selector functions in these contexts in no way shrinks the class of sets obtained. This observation enables one to use results from graph theory, in particular, from tournament theory, to pinpoint the advice complexity of selective sets.

Below we would like to make the link between commutative selector functions and directed graphs (in special cases, s-tournaments) more explicit and prove some results regarding this link.

Since our focus will be on self-contained associative functions, the following easy observation will be quite useful in the upcoming proofs. Note that the functions g spoken of in Proposition 3.1 may potentially be nontotal and/or multivalued.

PROPOSITION 3.1. *Let $B \subseteq \Sigma^*$ and let g be any commutative self-contained function that is total on B. Then the following are equivalent:*

1. *g is associative on B.*
2. $(\forall a, b, c \in B)[\textit{set-g}(a, \textit{set-g}(b, c)) = \textit{set-g}(b, \textit{set-g}(a, c)) = \textit{set-g}(c, \textit{set-g}(a, b))]$.
3. $(\forall a, b, c \in B : ||\{a, b, c\}|| = 3)[\textit{set-g}(a, \textit{set-g}(b, c)) = \textit{set-g}(b, \textit{set-g}(a, c)) = \textit{set-g}(c, \textit{set-g}(a, b))]$.

Proof. Let g be a commutative self-contained function that is total on B, $B \subseteq \Sigma^*$. Note that part 1 in light of g's commutativity implies part 2 and that part 2 implies part 3. We will complete the proof by showing that part 3 implies part 2 and that part 2 implies part 1.

Suppose that part 2 holds; that is, suppose $(\forall a, b, c \in B)[\textit{set-g}(a, \textit{set-g}(b, c)) = \textit{set-g}(b, \textit{set-g}(a, c)) = \textit{set-g}(c, \textit{set-g}(a, b))]$. Let $x, y, z \in B$. By our assumption we have $\textit{set-g}(x, \textit{set-g}(y, z)) = \textit{set-g}(z, \textit{set-g}(x, y))$. Since g is total on B and commutative we have $\textit{set-g}(z, \textit{set-g}(x, y)) = \textit{set-g}(\textit{set-g}(x, y), z)$ and thus $\textit{set-g}(x, \textit{set-g}(y, z)) = \textit{set-g}(\textit{set-g}(x, y), z)$. This by definition shows that g is associative.

Now suppose that part 3 holds; that is, suppose $(\forall a, b, c \in B : ||\{a, b, c\}|| = 3)[\textit{set-g}(a, \textit{set-g}(b, c)) = \textit{set-g}(b, \textit{set-g}(a, c)) = \textit{set-g}(c, \textit{set-g}(a, b))]$. Let $x, y, z \in B$. The only cases we need to consider are $||\{x, y, z\}|| = 1$ and $||\{x, y, z\}|| = 2$. If $||\{x, y, z\}|| = 1$, i.e., $x = y = z$, then $\textit{set-g}(x, \textit{set-g}(y, z)) = \textit{set-g}(y, \textit{set-g}(x, z)) = \textit{set-g}(z, \textit{set-g}(x, y)) = \{x\}$ since g is total on B and self-contained. If $||\{x, y, z\}|| = 2$, assume without loss of generality that $x = z$ and $x \neq y$. It follows that $\textit{set-g}(y, \textit{set-g}(x, z)) = \textit{set-g}(y, x)$ since g is total on B and self-contained, and $\textit{set-g}(y, x) = \textit{set-g}(x, y)$ since g is commutative. Observe that $\textit{set-g}(x, \textit{set-g}(y, z)) = \textit{set-g}(x, \textit{set-g}(y, x)) = \textit{set-g}(x, \textit{set-g}(x, y)) = \textit{set-g}(z, \textit{set-g}(x, y))$ since $x = z$ and g is commutative. It remains to show that $\textit{set-g}(z, \textit{set-g}(x, y)) = \textit{set-g}(y, \textit{set-g}(x, z))$, i.e., in the current setting, showing that $\textit{set-g}(x, \textit{set-g}(x, y)) = \textit{set-g}(x, y)$. Note that $\textit{set-g}(x, x) = \{x\}$ and $\textit{set-g}(x, y) \in \{\{x\}, \{y\}, \{x, y\}\}$ since g is total on B. If $\textit{set-g}(x, y) = \{x\}$, then $\textit{set-g}(x, \textit{set-g}(x, y)) = \textit{set-g}(x, y) = \{x\}$. If $\textit{set-g}(x, y) = \{y\}$, then $\textit{set-g}(x, \textit{set-g}(x, y)) = \textit{set-g}(x, y) = \{y\}$. If $\textit{set-g}(x, y) = \{x, y\}$, then $\textit{set-g}(x, \textit{set-g}(x, y)) = \textit{set-g}(x, y) = \{x, y\}$. We have shown that part 3 implies part 2. □

In particular note that, by Proposition 3.1, for any commutative self-contained function g that is total on some set B, g is associative on every one- or two-element set of strings from B, since item 3 of Proposition 3.1 is trivially satisfied when $||\{a, b, c\}|| < 3$.

Let f be a (single- or multivalued) commutative self-contained function, $f : B \times B \to B$ for a finite set B. Consider the digraph $G = (B, E)$, where $E = \{(x, y) \mid x, y \in B \wedge y \in \textit{set-f}(x, y)\}$. If f is total on B, then G is a complete digraph. If f is single-valued and total on B, then G is even an s-tournament. We will call G the f-induced digraph on B. If f is single-valued and total on B, then G will be called the f-induced s-tournament on B. Note that the functions f spoken of in Proposition 3.2 may be multivalued.

PROPOSITION 3.2. *Let B be a finite set and let f be a commutative self-contained function that is total on B. f is associative on B if and only if $E = \{(x, y) \mid x, y \in B \wedge y \in \textit{set-f}(x, y)\}$, the edge set of the f-induced digraph, is a transitive relation.*

Proof. Let B be a finite set and let f be a commutative self-contained function that is total on B.

We first prove the "only if" direction. So let f be associative on B, i.e., for all $x, y, z \in B$, $\textit{set-f}(x, \textit{set-f}(y, z)) = \textit{set-f}(\textit{set-f}(x, y), z)$. Suppose that E is not transitive. This means that there exist strings $a, b, c \in B$ such that $(a, b) \in E$ and $(b, c) \in E$ yet $(a, c) \notin E$. It follows that $a \neq b$ (since if $a = b$, then $(b, c) = (a, c)$), $b \neq c$ (since if $b = c$, then $(a, b) = (a, c)$), and $a \neq c$ (since if $a = c$, then $(a, a) = (a, c)$ and since f is total on B $(a, a) \in E$). So $a, b, c \in B$ are pairwise different strings such that $(a, b) \in E$, $(b, c) \in E$, and $(a, c) \notin E$. We will derive a contradiction by showing that f is not associative, i.e., $\textit{set-f}(\textit{set-f}(a, b), c) \neq \textit{set-f}(a, \textit{set-f}(b, c))$. We do so by showing $c \in \textit{set-f}(\textit{set-f}(a, b), c)$ and $c \notin \textit{set-f}(a, \textit{set-f}(b, c))$. We have $b \in \textit{set-f}(a, b)$ and $c \in \textit{set-f}(b, c)$, and thus $c \in \textit{set-f}(\textit{set-f}(a, b), c)$. On the other hand, $c \notin \textit{set-f}(a, b)$ since f is a self-contained function and $c \neq a$ and $c \neq b$. We also have $c \notin \textit{set-f}(a, c)$ since $(a, c) \notin E$. Since $\textit{set-f}(a, \textit{set-f}(b, c)) \subseteq \textit{set-f}(a, b) \cup \textit{set-f}(a, c)$ we have $c \notin \textit{set-f}(a, \textit{set-f}(b, c))$.

We now prove the "if" direction. So suppose that E is transitive. We have to show that f is associative on B, i.e., for any strings $a, b, c \in B$ we have to show $\textit{set-f}(a, \textit{set-f}(b, c)) = \textit{set-f}(\textit{set-f}(a, b), c)$. By Proposition 3.1 it suffices to show that, for all strings $a, b, c \in B$ such that $||\{a, b, c\}|| = 3$, $\textit{set-f}(a, \textit{set-f}(b, c)) =$

set-f$(b, $*set-f*$(a, c)) =$ *set-f*$(c, $*set-f*$(a, b))$. So suppose $a, b, c \in B$ are strings such that $||\{a, b, c\}|| = 3$.

We show *set-f*$(c, $*set-f*$(a, b)) \subseteq$ *set-f*$(a, $*set-f*$(b, c))$. (The proof of the other inclusions follows via variable renaming and commutativity.) Consider $w \in$ *set-f*$(c,$ *set-f*$(a, b))$. If $w = a$, we have $a \in$ *set-f*$(c, $*set-f*$(a, b))$, which implies, since $a \neq c$, $a \in$ *set-f*(a, b) and implies, since $a \notin \{b, c\}$, $a \in$ *set-f*$(c, a) =$ *set-f*(a, c). Since *set-f*$(a, b) \cap$ *set-f*$(a, c) \subseteq$ *set-f*$(a, $*set-f*$(b, c))$ it follows that $a \in$ *set-f*$(a, $*set-f*$(b, c))$. Similarly, if $w = b$, we have $b \in$ *set-f*$(c, $*set-f*$(a, b))$, which in turn implies (since $b \neq c$) $b \in$ *set-f*(a, b) and implies (since $b \notin \{a, c\}$ and $b \in$ *set-f*(a, b)) $b \in$ *set-f*$(c, b) =$ *set-f*(b, c). It follows that *set-f*$(a, b) \subseteq$ *set-f*$(a, $*set-f*$(b, c))$ and hence $b \in$ *set-f*$(a,$ *set-f*$(b, c))$. Finally suppose $w = c$, which yields $c \in$ *set-f*$(c, $*set-f*$(a, b))$. It follows (since $c \notin \{a, b\}$) that either (a) $a \in$ *set-f*(a, b) and $c \in$ *set-f*$(c, a) =$ *set-f*(a, c), or (b) $b \in$ *set-f*(a, b) and $c \in$ *set-f*$(c, b) =$ *set-f*(b, c). If (a) holds we have $(b, a) \in E$ and $(a, c) \in E$. By the transitivity of E it follows that also $(b, c) \in E$. By the definition of E this implies $c \in$ *set-f*(b, c), and thus $c \in$ *set-f*$(a, $*set-f*$(b, c))$. If (b) holds, we have $(a, b) \in E$ and $(b, c) \in E$. Hence also $(a, c) \in E$ since E is transitive. It follows that $c \in$ *set-f*(a, c), and thus $c \in$ *set-f*$(a, $*set-f*$(b, c))$. □

Tournaments such that their edge set is a transitive relation are called transitive tournaments. Transitive tournaments implicitly impose a linear order on their nodes. We will use the following result (which can be viewed as a tournament-language expression of the fact that each finite linear order has a minimal and also a maximal element).

PROPOSITION 3.3 (see [Moo68][3]). *The following statements are equivalent for s-tournaments $G = (V, E)$:*

1. *G is a transitive s-tournament.*
2. *G contains no directed cycles of length greater than* 1.
3. *Every induced (directed) subgraph $G' = (V', E')$, $V' \neq \emptyset$, of G contains a (distance-one) source node, i.e., a node $s \in V'$ such that for all $u \in V'$, $(s, u) \in E'$.*
4. *Every induced (directed) subgraph $G' = (V', E')$, $V' \neq \emptyset$, of G contains a (distance-one) target node, i.e., a node $t \in V'$ such that for all $u \in V'$, $(u, t) \in E'$.*

From Propositions 3.1, 3.2, and 3.3 we immediately obtain the following corollary.

COROLLARY 3.4. *Let f be a single-valued, commutative self-contained function that is total on $\{a, b, c\}$. f is not associative on $\{a, b, c\}$ if and only if both $||\{a, b, c\}|| = 3$ and the f-induced s-tournament on $\{a, b, c\}$ is a* 3*-cycle (plus three self-loops).*

A statement similar to that of Proposition 3.3 can also be made for complete digraphs. But in contrast to the linear order implicitly given by a transitive s-tournament, a transitive complete digraph gives a linear order on a partition of its set of nodes, while the nodes of each part of the partition are in some sense pairwise indistinguishable.

Without giving all the definitions, we mention that in graph theory textbooks (see [BJG00], for instance) one can find the fact that the strongly connected components of a general digraph form a partial order. If we consider a complete digraph, we immediately have that the strongly connected components form an order. The additional assumption of transitivity then yields that the strongly connected compo-

[3]Moon stated the result for tournaments, and so his statement lacks the "length greater than 1" condition in statement 2.

nents are actually strong cliques. However, for reasons of self-containment we state and prove that graph-theoretic result below in a way that suits our purposes best.

PROPOSITION 3.5. *The following statements are equivalent for complete digraphs* $G = (V, E)$*:*

1. E *is a transitive relation.*
2. *For every* $k \geq 1$*, if* $(a_1, a_2), (a_2, a_3), \ldots, (a_{k-1}, a_k), (a_k, a_1) \in E$*, then* $(a_i, a_j) \in E$ *for all* $1 \leq i, j \leq k$*. In other words, if* $a_1 \to a_2 \to \cdots \to a_{k-1} \to a_k \to a_1$ *is a directed cycle in* G*, then the nodes* $a_1, a_2, \ldots, a_k$ *form a strong clique.*
3. *Every induced (directed) subgraph* $G' = (V', E')$*,* $V' \neq \emptyset$*, of* G *contains a nonempty strong distance-one source-clique, i.e., an induced subgraph* $S = (V_S, E_S)$*,* $V_S \neq \emptyset$*, of* G' *having the following properties:*
   (a) S *is a maximal strong clique, i.e., for every* $v \in V' - V_S$*, the induced graph on the vertices* $V_S \cup \{v\}$ *is not a strong clique.*
   (b) *For all* $u \in V_S$ *and for all* $v \in V' - V_S$*,* $(u, v) \in E'$ *and* $(v, u) \notin E'$*.*
4. *Every induced (directed) subgraph* $G' = (V', E')$*,* $V' \neq \emptyset$*, of* G *contains a nonempty strong distance-one target-clique, i.e., an induced subgraph* $T = (V_T, E_T)$*,* $V_T \neq \emptyset$*, of* G' *having the following properties:*
   (a) T *is a maximal strong clique, i.e., for every* $v \in V' - V_T$*, the induced graph on the vertices* $V_T \cup \{v\}$ *is not a strong clique.*
   (b) *For all* $u \in V_T$ *and for all* $v \in V' - V_T$*,* $(u, v) \notin E'$ *and* $(v, u) \in E'$*.*

Proof. Let $G = (V, E)$ be a complete digraph. If $V = \emptyset$, all four conditions hold, so in this proof we henceforward assume $V \neq \emptyset$.

Statement 1 *implies statement* 2. Suppose that E is transitive. Transitivity implies that for any vertices u, v we have that the existence of a directed path from u to v implies $(u, v) \in E$. (This can easily be shown by induction on the length of the path connecting u and v.) Now consider a cycle $a_1 \to a_2 \to \cdots \to a_{k-1} \to a_k \to a_1$. Because for any vertices a_i, a_j there is a path from a_i to a_j, we have $(a_i, a_j) \in E$. Hence the nodes $a_1, a_2, \ldots, a_k$ form a strong clique.

Statement 2 *implies both statement* 3 *and statement* 4. Suppose that statement 2 holds for a complete digraph $G = (V, E)$. Let $G' = (V', E')$, $V' \neq \emptyset$, be an induced subgraph of G. There are strong cliques in G', because in a complete digraph every vertex forms a one-node strong clique. For the same reason, G' has no zero-node maximal strong cliques. We show that maximal strong cliques are disjoint. If two maximal strong cliques C and C' have a vertex in common, then there is a cycle connecting all vertices in C and C'. Then statement 2 implies that $C \cup C'$ forms a strong clique, contradicting the maximality of C and C'. Also, for two maximal strong cliques C and C' all edges between them lead from C to C', or all of them lead from C' to C, because otherwise there would be a cycle through all vertices of C and C', contradicting the maximality of C and C'. Since every vertex $v \in V'$ is in a maximal strong clique, the maximal strong cliques partition V'. Define a relation $\preceq$ on the maximal strong cliques by $C \preceq C'$ if and only if there are $a \in C$ and $a' \in C'$ with $(a, a') \in E$. Note that any two maximal strong cliques are comparable with respect to this relation, since G is a complete directed graph. Furthermore, the relation $\preceq$ is reflexive, antisymmetric, and transitive. Reflexivity and antisymmetry are obvious. Transitivity can be seen as follows: Suppose $C \preceq C'$ and $C' \preceq C''$ holds for three maximal strong cliques C, C', and C'' of G'. We have to show that then also $C \preceq C''$. If $C = C''$, then $C \preceq C''$ holds trivially since $\preceq$ is reflexive. If $C \neq C''$, then either $C \preceq C''$ or $C'' \preceq C$, but not both since $\preceq$ is antisymmetric. If $C \preceq C''$, we are done. If $C'' \preceq C$, then it indeed holds that there is a cycle through all vertices of $C \cup C' \cup C''$.

So, by statement 2, $C \cup C' \cup C''$ forms a strong clique, contradicting the maximality of C, C', and C''. Thus the transitivity of $\preceq$ is proven.

We have shown that the maximal strong cliques of G' are linearly ordered by $\preceq$. Let $C_{\min}$ be the minimum element, and let $C_{\max}$ be the maximum element in this order. ($C_{\min}$ and $C_{\max}$ do exist since G' is a (finite) graph and thus contains only a finite number of maximal strong cliques.) Then $C_{\min}$ witnesses statement 3 and $C_{\max}$ witnesses statement 4.

Statement 3 *implies statement* 1. Let $a, b, c \in V$ and suppose $(a, b) \in E$ and $(b, c) \in E$. We will show that also $(a, c) \in E$. Note that if $a = b$ or $a = c$ or $b = c$, then $(a, c) \in E$ is trivially satisfied. So let $||\{a, b, c\}|| = 3$ and consider the induced subgraph consisting of the three nodes a, b, and c—call it $G_{[a,b,c]} = (\{a, b, c\}, E_{abc})$. According to statement 3 there is a subgraph $S = (V_S, E_S)$, $V_S \subseteq \{a, b, c\}$, satisfying (a) S is a maximal strong clique and (b) for all $u \in V_S$ and for all $v \in \{a, b, c\} - V_S$, $(u, v) \in E_{abc}$ and $(v, u) \notin E_{abc}$. Hence, if $c \in V_S$, then $b \in V_S$, and if $b \in V_S$, then also $a \in V_S$. Since $V_S \neq \emptyset$, we can therefore conclude $a \in V_S$. It follows that $(a, c) \in E$.

Quite similarly to the proof that *statement* 3 *implies statement* 1, one can show that *statement* 4 *implies statement* 1. □

Combining Propositions 3.2 and 3.5 we get the following corollary.

COROLLARY 3.6. *Let f be a (potentially multivalued) commutative self-contained function that is total on $\{a, b, c\}$. f is not associative on $\{a, b, c\}$ if and only if both $||\{a, b, c\}|| = 3$ and the f-induced digraph on $\{a, b, c\}$ contains exactly one of the cycles $a \to b \to c \to a$ and $a \leftarrow b \leftarrow c \leftarrow a$.*

Proof. Let f be a multivalued, commutative self-contained function that is total on a set $\{a, b, c\}$. Let $G = (\{a, b, c\}, E)$ be the f-induced digraph on $\{a, b, c\}$.

Suppose that f is not associative on $\{a, b, c\}$. By the comment made immediately after the proof of Proposition 3.1, it follows that $||\{a, b, c\}|| = 3$. It follows from Proposition 3.2 that the edge set of the f-induced digraph is not a transitive relation and thus the f-induced digraph does not satisfy statement 2 of Proposition 3.5. It is not hard to see that this can only be the case if the f-induced digraph on $\{a, b, c\}$ contains exactly one of the cycles $a \to b \to c \to a$ and $a \leftarrow b \leftarrow c \leftarrow a$.

Now assume that $||\{a, b, c\}|| = 3$ and the f-induced digraph on $\{a, b, c\}$ contains exactly one of the cycles $a \to b \to c \to a$ and $a \leftarrow b \leftarrow c \leftarrow a$. By Proposition 3.5 we conclude that the edge set of the f-induced digraph is not transitive, which in turn, by Proposition 3.2, implies that f is not associative on $\{a, b, c\}$. □

Later in this paper, we will establish advice bounds in part by using the fact that associativity of a selector function (a restricted form of self-contained function) implies a linear structure on the underlying set.

4. Adding commutativity to associativity is free. Are all associatively P-selective sets commutatively, associatively P-selective? If they are, this will not only collapse two of our classes, but will also allow us to use results from the previous section in our study of associative selector functions. Unfortunately, Fact 2.3 (P-sel = C-P-sel) does *not* say that A-P-sel = CA-P-sel. The reason it does not guarantee this is that it is conceptually possible that the transformation denoted there by $(*)$ will, while gaining commutativity, destroy associativity. Theorem 4.1 and its proof establish that, for the case of $\mathrm{FP_t}$- and $\mathrm{NPSV_t}$-selector functions, this conceptual possibility is *not* a reality.

THEOREM 4.1.

1. A-P-sel = CA-P-sel.
2. A-NP-sel = CA-NP-sel.

Proof. As noted earlier, if B is P-selective (NP-selective) via a P-selector (NP-selector) function f, then the function $f'(x,y) = f(\min(x,y), \max(x,y))$ is a commutative P-selector (NP-selector) for B. We show that the above transformation from f to f' preserves associativity when f is a P-selector (NP-selector) function. Let $B \in$ A-P-sel ($B \in$ A-NP-sel) via an associative $\mathrm{FP_t}$-selector ($\mathrm{NPSV_t}$-selector) function f. It is clear that $f' \in \mathrm{FP_t}$ ($f' \in \mathrm{NPSV_t}$) and that f' is commutative. As mentioned above, f' is also a P-selector (NP-selector) for B. It remains to show that f' is associative.

Suppose that f' is not associative. According to Proposition 3.1 there exist three strings a, b, and c such that $||\{a,b,c\}|| = 3$ and $f'(a,f'(b,c)) = f'(b,f'(a,c)) = f'(c,f'(a,b))$ does not hold. Without loss of generality assume $a <_{\mathrm{lex}} b <_{\mathrm{lex}} c$. It follows from Corollary 3.4 that the f'-induced s-tournament on $\{a,b,c\}$ is a 3-cycle (plus three self-loops). Suppose that the cycle is directed from a to b, from b to c, and back to a. This implies that $f'(a,b) = b$, $f'(b,c) = c$, and $f'(a,c) = a$. However, due to the definition of f' this implies that $f(a,b) = f'(a,b) = b$, $f(b,c) = f'(b,c) = c$, and $f(a,c) = f'(a,c) = a$ which contradicts the associativity of f since $f(a,f(b,c)) = a$ yet $f(f(a,b),c) = c$. Similarly, if the cycle is directed from c to b, from b to a, and back to c, then $f'(a,b) = a$, $f'(b,c) = b$, and $f'(a,c) = c$. According to the definition of f' this implies $f(a,b) = f'(a,b) = a$, $f(b,c) = f'(b,c) = b$, and $f(a,c) = f'(a,c) = c$ which contradicts the associativity of f since $f(a,f(b,c)) = a$ yet $f(f(a,b),c) = c$. This shows that f' is associative. □

Similarly we have the following result.

THEOREM 4.2.

1. $\mathrm{A_\ell}$-P-sel = $\mathrm{A_\ell C}$-P-sel.
2. $\mathrm{A_\ell}$-NP-sel = $\mathrm{A_\ell C}$-NP-sel.

Proof. If a selector function f is associative at each length, then we may invoke the entire proof of Theorem 4.1, except only seeking a counterexample (to the "associativity at each length" of $f'(x,y) = f(\min(x,y), \max(x,y))$) among triples of strings all of the same length. As in that proof, if such a counterexample exists (among some three strings of the same length) it forms a 3-cycle (plus three self-loops) in the associated digraph. However, exactly as in the proof of Theorem 4.1, that contradicts the assumed associativity-at-each-length of f.[4] □

So the transformation from a selector f to a selector f' via $f'(x,y) = f(\min(x,y), \max(x,y))$ not only provides commutativity but also preserves associativity if f is total and single-valued. We mention that for this same case another standard transformation providing commutativity (namely, $f''(x,y) = \max(f(x,y), f(y,x))$) also has the associativity-preserving property.

However, the picture changes dramatically if one looks at multivalued selector

[4] Note that in the above proof, though commutativity of f' follows immediately from the definition of f', showing that f' is associative at each length relies on the fact that f is total at each length. However, the following strengthening of Theorem 4.2 clearly follows from the proof of Theorem 4.2: For every FP-selector (respectively, NPSV-selector) f for a set B that is associative at each length, f' is a commutative FP-selector (respectively, NPSV-selector) for B that is associative at each length n at which f is total.

However, we note in passing that one cannot strengthen Theorem 4.2 further to $\mathrm{A_\ell}$-FP-sel = $\mathrm{A_\ell C}$-FP-sel and $\mathrm{A_\ell}$-NPSV-sel = $\mathrm{A_\ell C}$-NPSV-sel by exploiting f'. A counterexample is the following: For strings a,b,c, $b <_{\mathrm{lex}} c <_{\mathrm{lex}} a$ all of the same length, let $\mathit{set}\text{-}f(a,b) = \mathit{set}\text{-}f(b,a) = \mathit{set}\text{-}f(a,c) = \mathit{set}\text{-}f(c,b) = \emptyset$ and $\mathit{set}\text{-}f(c,a) = \mathit{set}\text{-}f(b,c) = \{c\}$. One can easily check that f is associative on the three-element set $\{a,b,c\}$. But $f'(a,f'(b,c)) = c$ yet $f'(f'(a,b),c) = \emptyset$. Similarly to the counterexample showing that f' does not preserve associativity for multivalued functions (Proposition 4.3), one can extend the definition of f to cover all strings from Σ^*.

functions f, for instance, NPMV$_t$ selector functions. Though for the second transformation it is not even clear what is meant by $\max(f(x,y), f(y,x))$ and whether that is again a function from NPMV$_t$, it can be shown that the first transformation does *not* preserve associativity.

PROPOSITION 4.3. *There are total associative multivalued selector functions f such that the selector function f', defined by $f'(x,y) = f(\min(x,y), \max(x,y))$, is not associative.*

Proof. Set $a = \epsilon$, $b = 0$, and $c = 1$. Consider the total multivalued selector function f that has the following values on a, b, and c and thus induces the following values of f' on a, b, and c:

$$\begin{array}{lll|l} \textit{set-}f(a,b) = \{a\} & \textit{set-}f(b,a) = \{b\} & & \textit{set-}f'(a,b) = \textit{set-}f'(b,a) = \textit{set-}f(a,b) = \{a\} \\ \textit{set-}f(a,c) = \{a,c\} & \textit{set-}f(c,a) = \{c\} & & \textit{set-}f'(a,c) = \textit{set-}f'(c,a) = \textit{set-}f(a,c) = \{a,c\} \\ \textit{set-}f(b,c) = \{b,c\} & \textit{set-}f(c,b) = \{c\} & & \textit{set-}f'(b,c) = \textit{set-}f'(c,b) = \textit{set-}f(b,c) = \{b,c\} \end{array}$$

Furthermore, for each $x \in \{a,b,c\}$ and each $y \in \Sigma^* - \{a,b,c\}$, let $\textit{set-}f(x,y) = \textit{set-}f(y,x) = \{y\}$. And for $y,z \in \Sigma^* - \{a,b,c\}$ let $\textit{set-}f(y,z) = \textit{set-}f(z,y) = \{\max(y,z)\}$. Note that if $\{e,e'\} \not\subseteq \{a,b,c\}$, then we have $\textit{set-}f(e,e') = \textit{set-}f'(e,e')$.

It is not hard to verify that f is indeed associative, due to the way any non-$\{a,b,c\}$ argument "trumps" any $\{a,b,c\}$ argument, and regarding all-$\{a,b,c\}$ arguments we have

$$\begin{array}{l} \textit{set-}f(a, \textit{set-}f(b,c)) = \textit{set-}f(\textit{set-}f(a,b), c) = \{a,c\}, \\ \textit{set-}f(a, \textit{set-}f(c,b)) = \textit{set-}f(\textit{set-}f(a,c), b) = \{a,c\}, \\ \textit{set-}f(b, \textit{set-}f(a,c)) = \textit{set-}f(\textit{set-}f(b,a), c) = \{b,c\}, \\ \textit{set-}f(b, \textit{set-}f(c,a)) = \textit{set-}f(\textit{set-}f(b,c), a) = \{b,c\}, \\ \textit{set-}f(c, \textit{set-}f(a,b)) = \textit{set-}f(\textit{set-}f(c,a), b) = \{c\}, \\ \textit{set-}f(c, \textit{set-}f(b,a)) = \textit{set-}f(\textit{set-}f(c,b), a) = \{c\}. \end{array}$$

But f' is not associative since it holds that $\textit{set-}f'(a, \textit{set-}f'(c,b)) = \{a,c\}$ and $\textit{set-}f'(\textit{set-}f'(a,c), b) = \{a,b,c\}$. □

Observe that we have shown that under the mentioned transformation from f to f' gains in commutativity can lose associativity. However, for total multivalued functions there is another transformation that does preserve associativity while adding commutativity, namely, $\textit{set-}\widehat{f}(x,y) = \textit{set-}f(x,y) \cup \textit{set-}f(y,x)$ for all x and y. Theorem 4.4 establishes this.

THEOREM 4.4. A-NPMV$_t$-sel = CA-NPMV$_t$-sel.

Proof. It suffices to show A-NPMV$_t$-sel $\subseteq$ CA-NPMV$_t$-sel. Let $B \in$ A-NPMV$_t$-sel and let $f \in$ NPMV$_t$ be an associative selector for B. We will now show that $\widehat{f}$ is a commutative and associative NPMV$_t$-selector for B.

It is not hard to see that $\widehat{f}$ is commutative, that $\widehat{f} \in$ NPMV$_t$, and that indeed $\widehat{f}$ is an NPMV$_t$-selector for B. It remains to show that $\widehat{f}$ is associative. By Proposition 3.1 it suffices to show that $\widehat{f}$ is associative on every set $\{a,b,c\}$ such that $||\{a,b,c\}|| = 3$. By Corollary 3.6 it suffices to show that the $\widehat{f}$-induced graph on $\{a,b,c\}$, call it $G_{[a,b,c]} = (\{a,b,c\}, E_{abc})$, contains both the cycles $a \to b \to c \to a$ and $a \leftarrow b \leftarrow c \leftarrow a$ if it contains at least one of them.

Suppose $G_{[a,b,c]}$ contains $a \to b \to c \to a$ (the case where $G_{[a,b,c]}$ contains $a \leftarrow b \leftarrow c \leftarrow a$ is analogous under renaming so we do not separately cover it). We have to show that the other cycle is also in $G_{[a,b,c]}$, i.e., (a,c), (c,b), and (b,a) are in E_{abc}. For symmetry reasons, it suffices to show that $(a,c) \in E_{abc}$ or, equivalently, $c \in \textit{set-}\widehat{f}(a,c)$. Because $(a,b) \in E_{abc}$, we have $b \in \textit{set-}\widehat{f}(a,b)$ and thus $b \in \textit{set-}f(a,b)$

or $b \in set\text{-}f(b,a)$. Similarly, because $(b,c) \in E_{abc}$, we have $c \in set\text{-}\widehat{f}(b,c)$ and thus $c \in set\text{-}f(b,c)$ or $b \in set\text{-}f(c,b)$. We distinguish four cases.

Case 1 ($b \in set\text{-}f(a,b)$ and $c \in set\text{-}f(b,c)$):
This implies $c \in set\text{-}f(set\text{-}f(a,b),c)$. Since f is associative, we have that $c \in set\text{-}f(a, set\text{-}f(b,c))$. Since f is self-contained and thus $set\text{-}f(a, set\text{-}f(b,c)) \subseteq set\text{-}f(a,b) \cup set\text{-}f(a,c)$ we conclude $c \in set\text{-}f(b,c)$ and $c \in set\text{-}f(a,c)$. Therefore $c \in set\text{-}\widehat{f}(a,c)$.

Case 2 ($b \in set\text{-}f(a,b)$ and $c \in set\text{-}f(c,b)$):
This implies $c \in set\text{-}f(c, set\text{-}f(a,b))$. Since f is associative, we have that $c \in set\text{-}f(set\text{-}f(c,a),b)$. It follows that $c \in set\text{-}f(c,a)$ since f is self-contained. Therefore $c \in set\text{-}\widehat{f}(a,c)$.

Case 3 ($b \in set\text{-}f(b,a)$ and $c \in set\text{-}f(b,c)$):
This implies $c \in set\text{-}f(set\text{-}f(b,a),c)$. Since f is associative, we have that $c \in set\text{-}f(b, set\text{-}f(a,c))$. Again, it follows that $c \in set\text{-}f(a,c)$ since f is self-contained. Therefore $c \in set\text{-}\widehat{f}(a,c)$.

Case 4 ($b \in set\text{-}f(b,a)$ and $c \in set\text{-}f(c,b)$):
This implies $c \in set\text{-}f(c, set\text{-}f(b,a))$. Since f is associative, we have that $c \in set\text{-}f(set\text{-}f(c,b),a)$. It follows that $c \in set\text{-}f(c,b)$ since f is self-contained and thus $c \in set\text{-}f(c,a)$. Therefore $c \in set\text{-}\widehat{f}(a,c)$.

It follows that $(a,c) \in E_{abc}$. This completes the proof that $\widehat{f}$ is associative. □

We also have the following.

THEOREM 4.5. $\mathrm{A}_\ell\text{-NPMV}_\mathrm{t}\text{-sel} = \mathrm{A}_\ell\text{C-NPMV}_\mathrm{t}\text{-sel}$.

The proof of this theorem proceeds in analogy to the proof of Theorem 4.4, except that here we focus on strings of equal length.[5]

Finally we turn to partial selector functions. The definition of selector functions allows a partial selector function for a set B to be undefined only when both inputs are not contained in B. However, the additional requirement of associativity forces any selector function for a nonempty set to be outright total.

PROPOSITION 4.6.

1. *If a (single- or multivalued) selector function f for a set $B \neq \emptyset$ is associative, then f is a total function.*
2. *If a (single- or multivalued) selector function f for a set B is associative at each length, then $f(x,y)$ is defined for all x and y such that both $|x| = |y|$ and $B^{=|x|} \neq \emptyset$.*

Proof. We first prove part 1 of the proposition. Let f be an associative selector function for a set $B \neq \emptyset$. For any inputs x and y such that $\{x,y\} \cap B \neq \emptyset$, $f(x,y)$ is not undefined (equivalently, $set\text{-}f(x,y) \neq \emptyset$), due to the definition of selectivity. Assume $x, y \notin B$ and $f(x,y)$ is undefined, i.e., $set\text{-}f(x,y) = \emptyset$. Consider a third string $z \in B$. Such a string exists since $B \neq \emptyset$. Note that $set\text{-}f(x,z) = \{z\}$ and $set\text{-}f(y,z) = \{z\}$ according to the definition of a selector function. So $set\text{-}f(x, set\text{-}f(y,z)) = \{z\}$

[5]Comments similar to those of footnote 4 apply here. Namely, the following strengthening of Theorem 4.5 clearly follows from the proof of Theorem 4.5: For every NPMV-selector f for a set B that is associative at each length, $\widehat{f}$ is a commutative NPMV-selector for B that is associative at each length n at which f is total.

However, Theorem 4.5 cannot be strengthened to A_ℓ-NPMV-sel = A_ℓC-NPMV-sel using $\widehat{f}$. A counterexample is the following: For strings a, b, c all of the same length, let $set\text{-}f(a,b) = set\text{-}f(b,a) = set\text{-}f(a,c) = set\text{-}f(c,b) = \emptyset$ and $set\text{-}f(c,a) = set\text{-}f(b,c) = \{c\}$. One can easily check that f is associative on the three-element set $\{a,b,c\}$. But $set\text{-}\widehat{f}(a, set\text{-}\widehat{f}(b,c)) = \{c\}$ yet $set\text{-}\widehat{f}(set\text{-}\widehat{f}(a,b),c) = \emptyset$. Similarly to the counterexample showing that f' does not preserve associativity for multivalued functions (Proposition 4.3), one can extend the definition of f to cover all strings from Σ^*.

yet (since $\textit{set-}f(x,y) = \emptyset$) $\textit{set-}f(\textit{set-}f(x,y),z) = \emptyset$. This contradicts the associativity of f.

The proof of part 2 is very similar to the proof of part 1. □

COROLLARY 4.7.

1. A-FP-sel = A-$\mathrm{FP_t}$-sel = CA-$\mathrm{FP_t}$-sel. *(That is,* A-FP-sel = A-P-sel = CA-P-sel.*)*
2. *Every set* $B \in \mathrm{A_\ell}$-FP-sel *has a commutative* FP*-selector that is total at each length* n *for which* $B^{=n} \neq \emptyset$ *and associative at each length* n *for which* $B^{=n} \neq \emptyset$.
3. A-NPSV-sel = A-$\mathrm{NPSV_t}$-sel = CA-$\mathrm{NPSV_t}$-sel. *(That is,* A-NPSV-sel = A-NP-sel = CA-NP-sel.*)*
4. *Every set* $B \in \mathrm{A_\ell}$-NPSV-sel *has a commutative* NPSV*-selector that is total at each length* n *for which* $B^{=n} \neq \emptyset$ *and associative at each length* n *for which* $B^{=n} \neq \emptyset$.
5. A-NPMV-sel = A-$\mathrm{NPMV_t}$-sel = CA-$\mathrm{NPMV_t}$-sel.
6. *Every set* $B \in \mathrm{A_\ell}$-NPMV-sel *has a commutative* NPMV*-selector that is total at each length* n *for which* $B^{=n} \neq \emptyset$ *and associative at each length* n *for which* $B^{=n} \neq \emptyset$.

Parts 1 and 3 follow directly from Proposition 4.6 and Theorem 4.1. Note that $\emptyset$ is trivially contained in CA-P-sel. Parts 2 and 4 follow implicitly from Proposition 4.6 and the proof of Theorem 4.2 (note footnote 4). Part 5 follows directly from Proposition 4.6 and Theorem 4.4. Part 6 again follows implicitly from Proposition 4.6 and the proof of Theorem 4.5 (note footnote 5).

5. Associativity drops advice complexity to linear. Ko showed the following result.

THEOREM 5.1 (see [Ko83]). P-sel $\subseteq \mathrm{P}/\mathcal{O}(n^2)$.

That is, deterministic advice interpreters given quadratic advice accept the P-selective sets. It is also known that nondeterministic advice interpreters with linear advice accept the P-selective sets, and this is optimal.

THEOREM 5.2 (see [HT96]; see also [HNP98]). P-sel $\subseteq \mathrm{NP}/n{+}1$ *but* P-sel $\not\subseteq \mathrm{NP}/n$.

It is natural to wonder whether the "P" advice interpretation of Theorem 5.1 can be unified with the linear advice amount of Theorem 5.2 to obtain the very strong claim P-sel $\subseteq \mathrm{P}/\mathcal{O}(n)$. Currently, no proof of this claim is known, researchers studying selectivity generally doubt that it holds, and there is some very recent relativized evidence in harmony with that doubt [Tha03]. However, proving the claim false seems unlikely in the immediate future, since by Theorem 5.2 any such proof would implicitly prove P ≠ NP.

Nonetheless, the following result shows that all associatively P-selective sets *are* accepted by deterministic advice interpreters with linear advice.

THEOREM 5.3. $\mathrm{A_\ell}$-FP-sel $\subsetneq \mathrm{P}/n{+}1$.

$(\mathrm{P}/1) - \mathrm{NPMV\text{-}sel} \neq \emptyset$ (see the comments in the proof of Theorem 5.3 below regarding why this holds). In light of this inequality, Theorem 5.3 supports the following corollary.

COROLLARY 5.4.

1. A-P-sel $\subsetneq \mathrm{P}/n{+}1$.
2. $\mathrm{A_\ell}$-P-sel $\subsetneq \mathrm{P}/n{+}1$.

Proof of Theorem 5.3. Let $B \in \mathrm{A_\ell}$-FP-sel. Let f be a commutative FP-selector for B that (a) is total at each length n such that $B^{=n} \neq \emptyset$ and that (b) is associative at

each length n such that $B^{=n} \neq \emptyset$. By Corollary 4.7 we know that such a commutative FP-selector for B must exist.

Let $n \in \mathbb{N}$ such that $B^{=n} \neq \emptyset$. Consider the directed graph $G_n = (B^{=n}, E_n)$, where $E_n = \{(x, y) \mid x, y \in B^{=n} \wedge f(x, y) = y\}$. G_n is a nonempty s-tournament.

Since f is a commutative selector function that is total and associative on $B^{=n}$, it follows from Proposition 3.2 that E_n is a transitive relation. So G_n is a transitive s-tournament, which by Proposition 3.3 implies that G_n contains a source node s_n. Note that $x \in B^{=n}$ implies $f(x, s_n) = x$ and $x \in \Sigma^n - B^{=n}$ implies $f(x, s_n) = s_n$. In other words, for all $n \in \mathbb{N}$, if $B^{=n} \neq \emptyset$ then there exists a string $s_n \in B^{=n}$ such that for all $x \in \Sigma^n$,

$$x \in B \iff f(x, s_n) = x.$$

Define, for all $n \in \mathbb{N}$,

$$h(1^n) = \begin{cases} 1s_n & \text{if } B^{=n} \neq \emptyset, \\ 0^{n+1} & \text{otherwise.} \end{cases}$$

Note that h on any input 1^n outputs a string of length exactly $n + 1$. Let $L = \{\langle x, 1y\rangle \mid |x| = |y| \wedge f(x, y) = x\}$. Clearly, $L \in \mathrm{P}$. Due to the above construction we also have, for all $x \in \Sigma^*$,

$$x \in B \iff \langle x, h(1^{|x|})\rangle \in L.$$

This shows that $B \in \mathrm{P}/n{+}1$.

It remains to show that $\mathrm{P}/n{+}1 - \mathrm{A}_\ell\text{-FP-sel} \neq \emptyset$. In fact, it clearly holds even that $(\mathrm{P}/1) - \text{NPMV-sel} \neq \emptyset$. The proof of this claim consists of a diagonalization against all self-contained NPMV functions. Let $f_1, f_2, \ldots$ be an enumeration (not necessarily effective, though in fact there do exist such effective enumerations) of all self-contained NPMV functions. Set

$$\begin{aligned} B = &\left\{0^{2i-1} \mid i \in \mathbb{N} \wedge \mathit{set}\text{-}f_i(0^{2i-2}, 0^{2i-1}) = \{0^{2i-2}\}\right\} \\ &\cup \left\{0^{2i-2} \mid i \in \mathbb{N} \wedge \mathit{set}\text{-}f_i(0^{2i-2}, 0^{2i-1}) \neq \{0^{2i-2}\}\right\}. \end{aligned}$$

B is not NPMV-selective, yet $B \in \mathrm{P}/1$. □

Note that for the correctness of the proof of Theorem 5.3 it actually suffices for f to be a partial FP-selector that is weakly associative at each length. f being a selector for B ensures that G_n is an s-tournament since partial selectors for B can be undefined only if both inputs are not in B. The weak associativity at each length of f yields that E_n is transitive. And finally, the above-defined set L remains a P set even under those weaker assumptions.

We note also that, more generally than Theorem 5.3, if a set A and a (not necessarily commutative) P-selector f for it have the property that at each nonempty length m there is some string $y \in A^{=m}$ at that length—or even at some length linearly related to that length—such that $f(y, z) = z$ or $f(z, y) = z$ for each $z \in A^{=m}$, then A is in $\mathrm{P}/\mathcal{O}(n)$. However, among ways of ensuring that this condition is met, we feel that associativity is a particularly natural and well-structured property to study.

The number of advice bits in Theorem 5.3 and Corollary 5.4, $n + 1$, is optimal.

THEOREM 5.5. $\text{A-P-sel} \not\subseteq \mathrm{P}/n$.

Proof. Hemaspaandra and Torenvliet [HT96] construct a P-selective set, which we will call B, such that $B \not\subseteq \mathrm{P}/n$. We will show that their set is A-selective. (We will

not repeat their proof that $B \not\subseteq \mathrm{P}/n$, since they already proved that.) Hemaspaandra and Torenvliet do not completely specify the P-selector function for B, though they make it clear that B is P-selective. In fact, it is easy to see that there are ways of completing the specification of their P-selector function so that it is not associative. However, our task is merely to prove that there is at least one way to complete their P-selector function so as to make it associative. We now do so.

The set B constructed by Hemaspaandra and Torenvliet satisfies the following properties:

1. $B \in \mathrm{DTIME}[2^{2^n}]$.
2. For all strings x, if $x \in B$, then $|x| \in \mathcal{L}$, where $\ell_0 = 2$, $\ell_{i+1} = 2^{2^{\ell_i}}$ for all $i \geq 0$, and $\mathcal{L} = \{\ell_0, \ell_1, \ell_2, \dots\}$.
3. For all strings x and y such that $|x| = |y|$, if $x \leq_{\mathrm{lex}} y$, then $\chi_B(x) \leq \chi_B(y)$, where χ_B denotes the characteristic function of B.

Let f be the following function:

$$f(x,y) = \begin{cases} x & \text{if } |x| \in \mathcal{L} \text{ and } |y| \notin \mathcal{L}, \\ y & \text{if } |x| \notin \mathcal{L} \text{ and } |y| \in \mathcal{L}, \\ x & \text{if } |x|, |y| \in \mathcal{L},\ |x| < |y|, \text{ and } x \in B, \\ y & \text{if } |x|, |y| \in \mathcal{L},\ |x| < |y|, \text{ and } x \notin B, \\ y & \text{if } |x|, |y| \in \mathcal{L},\ |x| > |y|, \text{ and } y \in B, \\ x & \text{if } |x|, |y| \in \mathcal{L},\ |x| > |y|, \text{ and } y \notin B, \\ \max(x,y) & \text{otherwise.} \end{cases}$$

Note that the "otherwise" case is reached exactly when

$$(|x| \notin \mathcal{L} \text{ and } |y| \notin \mathcal{L}) \text{ or } |x| = |y|.$$

In light of the above properties of B it is not hard to verify that f is polynomial-time computable. The key thing to note to see this is that when $|x|, |y| \in \mathcal{L}$ and $|x| \neq |y|$, then in that case the huge gaps in $\mathcal{L}$ and the fact that $B \in \mathrm{DTIME}[2^{2^n}]$ allow us to brute-force test $\min(x,y) \in B$ in time polynomial in $|x| + |y|$. Also (the polynomial-time computable function) f is clearly a P-selector for B.

What remains to show is that f is associative. From its definition, f is clearly commutative. So by Proposition 3.1, it suffices to show that for all $x, y, z \in \Sigma^*$, $||\{x,y,z\}|| = 3$, it holds that $f(x, f(y,z)) = f(y, f(x,z)) = f(z, f(x,y))$. So let a, b, and c be three strings such that $||\{a,b,c\}|| = 3$. Assume without loss of generality that $a <_{\mathrm{lex}} b <_{\mathrm{lex}} c$. Clearly it will also hold that $|a| \leq |b| \leq |c|$.

Case 1 ($|a|, |b|, |c| \notin \mathcal{L}$):
Applying the definition of f, we see that $f(a, f(b,c)) = f(b, f(a,c)) = f(c, f(a,b)) = \max(a,b,c)$.

Case 2 (the length of exactly one string from $\{a,b,c\}$ is in $\mathcal{L}$):
Let $w \in \{a,b,c\}$ be the unique string among a, b, and c such that $|w| \in \mathcal{L}$. Let v_1 and v_2 be the two strings in $\{a,b,c\}$ such that $|v_1|, |v_2| \notin \mathcal{L}$. So $f(w, v_1) = w$, $f(w, v_2) = w$, and consequently, since f is a self-contained function, $f(w, f(v_1, v_2)) = f(v_1, f(w, v_2)) = f(v_2, f(w, v_1)) = w$. It follows that $f(a, f(b,c)) = f(b, f(a,c)) = f(c, f(a,b)) = w$.

Case 3 (the length of exactly two strings from $\{a,b,c\}$ is in $\mathcal{L}$):
Let $v \in \{a,b,c\}$ be the unique string among a, b, and c such that $|v| \notin \mathcal{L}$. Let w_1 and w_2 be the two strings in $\{a,b,c\}$ such that $|w_1|, |w_2| \in \mathcal{L}$. So $f(v, w_1) = w_1$, $f(v, w_2) = w_2$, and consequently $f(v, f(w_1, w_2)) = f(w_1, f(v, w_2)) = f(w_2, f(v, w_1)) = f(w_1, w_2)$. It follows that $f(a, f(b,c)) = f(b, f(a,c)) = f(c, f(a,b)) = f(w_1, w_2)$.

Case 4 ($|a|, |b|, |c| \in \mathcal{L}$):

Case 4.1 ($\|\{|a|, |b|, |c|\}\| = 1$):

So $|a| = |b| = |c|$ and thus, by the definition of f, we have $f(a, f(b, c)) = f(b, f(a, c)) = f(c, f(a, b)) = \max(a, b, c)$.

Case 4.2 ($\|\{|a|, |b|, |c|\}\| = 2$):

So, as $|a| \leq |b| \leq |c|$, it holds that $|a| = |b|$ or $|b| = |c|$, yet not all three strings have the same length.

Case 4.2.1 ($|a| = |b|$ and $|b| < |c|$):

Note that $f(a, b) = \max(a, b) = b$,

$$f(a,c) = \begin{cases} a & \text{if } a \in B, \\ c & \text{otherwise,} \end{cases} \qquad \text{and} \qquad f(b,c) = \begin{cases} b & \text{if } b \in B, \\ c & \text{otherwise.} \end{cases}$$

Recall that $a <_{\text{lex}} b <_{\text{lex}} c$. Due to the third property of B quoted earlier in this proof, we thus have $a \in B \implies b \in B$ or, equivalently, $b \notin B \implies a \notin B$. So

$$f(a, f(b, c)) = f(b, f(a, c)) = f(c, f(a, b)) = \begin{cases} b & \text{if } b \in B, \\ c & \text{otherwise.} \end{cases}$$

Case 4.2.2 ($|a| < |b|$ and $|b| = |c|$):

Recall that $a <_{\text{lex}} b <_{\text{lex}} c$. So $f(b, c) = \max(b, c) = c$,

$$f(a,b) = \begin{cases} a & \text{if } a \in B, \\ b & \text{otherwise,} \end{cases} \qquad \text{and} \qquad f(a,c) = \begin{cases} a & \text{if } a \in B, \\ c & \text{otherwise.} \end{cases}$$

It follows that

$$f(a, f(b, c)) = f(b, f(a, c)) = f(c, f(a, b)) = \begin{cases} a & \text{if } a \in B, \\ c & \text{otherwise.} \end{cases}$$

Case 4.3 ($\|\{|a|, |b|, |c|\}\| = 3$):

So we have $|a|, |b|, |c| \in \mathcal{L}$ and $|a| < |b| < |c|$. Thus

$$f(a,b) = \begin{cases} a & \text{if } a \in B, \\ b & \text{otherwise,} \end{cases} \qquad f(a,c) = \begin{cases} a & \text{if } a \in B, \\ c & \text{otherwise,} \end{cases}$$

$$\text{and} \qquad f(b,c) = \begin{cases} b & \text{if } b \in B, \\ c & \text{otherwise.} \end{cases}$$

It follows that

$$f(a, f(b, c)) = f(b, f(a, c)) = f(c, f(a, b)) = \begin{cases} a & \text{if } a \in B, \\ b & \text{if } a \notin B \text{ and } b \in B, \\ c & \text{otherwise.} \end{cases}$$

This completes the proof of the claim that f is associative. □

COROLLARY 5.6.

1. A_ℓ-P-sel $\not\subseteq$ P/n.
2. A_ℓ-FP-sel $\not\subseteq$ P/n.

In fact, for any recursive function g, A-P-sel $\not\subseteq$ DTIME$[g(n)]/n$. The reason for this is that we may take the P-selective set that Hemaspaandra and Torenvliet [HT96] prove is not in DTIME$[g(n)]/n$ and, just as in the proof of Theorem 5.5, may complete

its P-selector function in a manner that achieves associativity. From this observation, it follows that the linear advice bound for associatively NP-selective sets and also the linear advice bound for associatively NPMV-selective sets that will be proven as Theorem 5.9 and Corollary 5.10 are optimal.

COROLLARY 5.7.

1. A-NP-sel $\not\subseteq$ (NP $\cap$ coNP)/n.
2. A_ℓ-NP-sel $\not\subseteq$ (NP $\cap$ coNP)/n.
3. A_ℓ-NPSV-sel $\not\subseteq$ (NP $\cap$ coNP)/n.
4. A-NPMV$_t$-sel $\not\subseteq$ NP/n.
5. A_ℓ-NPMV$_t$-sel $\not\subseteq$ NP/n.
6. A_ℓ-NPMV-sel $\not\subseteq$ NP/n.

So we have established an n-bit lower bound for the advice complexity of (associatively) NP- and NPMV-selective sets. The following upper bounds on the amount of advice needed to (nondeterministically) accept NP- and NPMV-selective sets are known.

THEOREM 5.8.

1. [HHN$^+$95] NP-sel $\subseteq$ (NP $\cap$ coNP)/poly.
2. [HNOS96b] NPSV-sel $\subseteq$ NP/poly $\cap$ coNP/poly *and* NPSV-sel $\cap$ NP $\subseteq$ (NP $\cap$ coNP)/poly.
3. [HNOS96b] NPMV$_t$-sel $\subseteq$ NP/poly $\cap$ coNP/poly.
4. [HNOS96b] NPMV-sel $\subseteq$ NP/poly.

We note that the results we have stated that involve complexity classes relativize. In particular, one can relativize them by any particular NP $\cap$ coNP set, and so regarding associativity and the NP-selective sets (equivalently, the NPSV$_t$-selective sets and, equivalently, the $\mathrm{FP}_t^{\mathrm{NP}\cap\mathrm{coNP}}$-selective sets and, equivalently, the $\mathrm{FP}^{\mathrm{NP}\cap\mathrm{coNP}}$-selective sets; see [HNOS96b, HHN$^+$95] for definitions and discussion; note that $\mathrm{FP}_t^{\mathrm{NP}\cap\mathrm{coNP}}$-sel $= \mathrm{FP}^{\mathrm{NP}\cap\mathrm{coNP}}$-sel for basically the same reason that FP$_t$-sel = FP-sel), it follows from our results that all length-associatively NP-selective sets are in (NP $\cap$ coNP)/$n+1$.

THEOREM 5.9.

1. A_ℓ-NP-sel $\subsetneq$ (NP $\cap$ coNP)/n+1.
2. A_ℓ-NPMV-sel $\subsetneq$ NP/n+1 $\cap$ coNP/n+1.

Proof. Regarding the strictness ($\subsetneq$ rather than $\subseteq$) of both parts, this follows from the easy fact (noted in the proof of Theorem 5.3) that (P/1) $-$ NPMV-sel $\neq \emptyset$. The containment claims of both parts remain to be proven.

We now prove the containment part of item 1 of the theorem, i.e., that A_ℓ-NP-sel $\subseteq$ (NP $\cap$ coNP)/n+1. Let $B \in A_\ell$-NP-sel. It follows from Theorem 4.2 that there exists a commutative NPSV$_t$-selector f for B that is associative at each length.

One can now invoke the entire containment part of the proof of Theorem 5.3, i.e., the proof for the claim A_ℓ-P-sel $\subseteq$ P/n+1 (this is actually that proof weakened to the case described by part 2 of Corollary 5.4), with the obvious change from P-selector to NPSV$_t$-selector, and thus resulting in the fact that the set L defined in the proof of Theorem 5.3 is now in NP $\cap$ coNP. To see the latter, simply note that $L = \{\langle x, 1y\rangle \mid |x| = |y| \wedge f(x,y) = x\}$ can also be written as $L = \{\langle x, 1y\rangle \mid |x| = |y| \wedge (x = y \vee f(x,y) \neq y)\}$ since f is a total NPSV-selector.

We now show the containment part of item 2, i.e., that A_ℓ-NPMV-sel $\subseteq$ NP/n+1 $\cap$ coNP/n+1. Let $B \in A_\ell$-NPMV-sel. It follows from Corollary 4.7 that there exists a commutative NPMV-selector f for B that is total at each length at which B is nonempty, and that is associative at each length at which B is nonempty.

Let $n \in \mathbb{N}$ be such that $B^{=n} \neq \emptyset$. Consider the digraph $G_n = (B^{=n}, E_n)$, where $E_n = \{(x,y) \mid x, y \in B^{=n} \wedge y \in \mathit{set}\text{-}f(x,y)\}$. By the definition of NPMV-selectivity

(see Definition 2.1), f must be defined for all $x, y \in B^{=n}$. So G_n is a nonempty complete digraph. In fact, note even that, from Proposition 4.6, f is defined for all $x, y \in \Sigma^n$. Since f is a commutative selector function that is total and associative on $B^{=n}$, it follows from Proposition 3.2 that E_n is a transitive relation. So it follows from Proposition 3.5 that G_n contains a nonempty strong distance-one source-clique, i.e., an induced subgraph $S = (V_S, E_S)$, $V_S \neq \emptyset$, having the following properties:

1. S is a maximal strong clique.
2. For all $u \in V_S$ and for all $v \in B^{=n} - V_S$, $(u, v) \in E_n$ and $(v, u) \notin E_n$.

Let u_n denote the lexicographically smallest string in V_S. (This is just for specificity. Any string from V_S would function the same way.) Note that $x \in B^{=n}$ implies $x \in \mathit{set}\text{-}f(u_n, x)$, whereas $x \in \Sigma^n - B^{=n}$ implies $x \notin \mathit{set}\text{-}f(u_n, x)$. Hence, for all $n \in \mathbb{N}$ such that $B^{=n} \neq \emptyset$ there exists a string $u_n \in B^{=n}$ such that for all $x \in \Sigma^n$,

$$x \in B \iff x \in \mathit{set}\text{-}f(u_n, x).$$

Define, for all $n \in \mathbb{N}$,

$$h(1^n) = \begin{cases} 1u_n & \text{if } B^{=n} \neq \emptyset, \\ 0^{n+1} & \text{otherwise.} \end{cases}$$

Note that h on any input 1^n outputs a string of length exactly $n+1$. Let $L = \{\langle x, 1y\rangle \mid |x| = |y| \wedge x \in \mathit{set}\text{-}f(y, x)\}$. Clearly, $L \in \mathrm{NP}$. Due to the above construction we also have, for all $x \in \Sigma^*$,

$$x \in B \iff \langle x, h(1^{|x|})\rangle \in L.$$

This shows that $B \in \mathrm{NP}/n{+}1$.

To see that $B \in \mathrm{coNP}/n{+}1$ we have to repeat the argument on $\Sigma^n - B^{=n}$. Let $n \in \mathbb{N}$ such that both $B^{=n} \neq \emptyset$ and $\Sigma^n - B^{=n} \neq \emptyset$. (If $B^{=n} = \emptyset$ or $\Sigma^n - B^{=n} = \emptyset$, then we at this length easily have a $\mathrm{coNP}/n{+}1$ algorithm for that case and will signal such cases explicitly in the advice function that we will soon build.) Consider the digraph $G'_n = (\Sigma^n - B^{=n}, E'_n)$, where $E'_n = \{(x, y) \mid x, y \in \Sigma^n - B^{=n} \wedge y \in \mathit{set}\text{-}f(x, y)\}$. G'_n is a nonempty complete digraph. (G'_n is nonempty since $\Sigma^n - B^{=n} \neq \emptyset$ and G'_n is a complete digraph since f is total at length n due to $B^{=n} \neq \emptyset$ and Proposition 4.6.) Since f is a commutative selector function that is total and associative on $\Sigma^n - B^{=n}$ it follows from Proposition 3.2 that E'_n is a transitive relation. Hence it follows from Proposition 3.5 that G'_n contains a nonempty strong distance-one target-clique, i.e., an induced subgraph $T = (V_T, E_T)$, $V_T \neq \emptyset$, having the following properties:

1. T is a maximal strong clique.
2. For all $u \in V_T$ and for all $v \in (\Sigma^n - B^{=n}) - V_T$, $(u, v) \notin E'_n$ and $(v, u) \in E'_n$.

Let v_n denote the lexicographically largest string in V_T. (This is just for specificity. Any string from V_T would function the same way.) It follows that for $n \in \mathbb{N}$, if $B^{=n} \neq \emptyset$ and $\Sigma^n - B^{=n} \neq \emptyset$, then there exists a string $v_n \in \Sigma^n - B^{=n}$ such that for all $x \in \Sigma^n$,

$$x \in B \iff v_n \notin \mathit{set}\text{-}f(x, v_n).$$

Define, for all $n \in \mathbb{N}$,

$$h'(1^n) = \begin{cases} 0^{n+1} & \text{if } B^{=n} = \emptyset, \\ 01^n & \text{if } B^{=n} = \Sigma^n, \\ 1v_n & \text{if } B^{=n} \neq \emptyset \text{ and } B^{=n} \neq \Sigma^n. \end{cases}$$

Note that h' on any input 1^n outputs a string of length exactly $n+1$. Let $L' = \{\langle x, 1y\rangle \mid |x| = |y| \wedge y \notin \mathit{set}\text{-}f(x,y)\} \cup \{\langle x, 01^{|x|}\rangle \mid x \in \Sigma^*\}$. Clearly, $L' \in \mathrm{coNP}$. Due to the above construction we also have, for all $x \in \Sigma^*$,

$$x \in B \iff \langle x, h'(1^{|x|})\rangle \in L'.$$

This shows that $B \in \mathrm{coNP}/n{+}1$. □

Since, as noted earlier, $(\mathrm{P}/1) - \mathrm{NPMV\text{-}sel} \neq \emptyset$, Theorem 5.9 yields the following corollary.

COROLLARY 5.10.

1. $\mathrm{A\text{-}NP\text{-}sel} \subsetneq (\mathrm{NP} \cap \mathrm{coNP})/n{+}1$.
2. $\mathrm{A}_\ell\mathrm{\text{-}NPSV\text{-}sel} \subsetneq \mathrm{NP}/n{+}1 \cap \mathrm{coNP}/n{+}1$.
3. $\mathrm{A\text{-}NPMV}_\mathrm{t}\mathrm{\text{-}sel} \subsetneq \mathrm{NP}/n{+}1 \cap \mathrm{coNP}/n{+}1$.
4. $\mathrm{A}_\ell\mathrm{\text{-}NPMV}_\mathrm{t}\mathrm{\text{-}sel} \subsetneq \mathrm{NP}/n{+}1 \cap \mathrm{coNP}/n{+}1$.

We mention in passing that—since $\log_2((2^{n+1}-1)+1) = n+1$ and thus one can include in the advice-seeking s-tournament all lengths up to the current one—the $n+1$ advice bounds in part 1 of Corollaries 5.4 and 5.10 even hold in the "strong advice" model of Ko, Balcázar, and Schöning [Ko87, BS92] in which advice must work for all strings up to and including a given length (see also the discussion in [HT96]). The $n+1$ advice bound in the "$\mathrm{NP}/n+1$" of part 3 of Corollary 5.10 for the same reason holds also in the "strong advice" model. Since the "$\mathrm{coNP}/n+1$" part of the proof of part 2 of Theorem 5.9 had *two* special-meaning advice cases, at first it might seem that regarding the "$\mathrm{coNP}/n+1$" of part 3 of Corollary 5.10 $\lceil \log_2((2^{n+1}-1)+2)\rceil = n+2$ advice bits, rather than $n+1$ advice bits, are needed in the "strong advice" model. However, note that we can hardcode into our machine the issue of whether some one string, say ϵ, is in our $\mathrm{A\text{-}NPMV}_\mathrm{t}$-sel set, and then can leave ϵ out of all the tournaments and always handle it directly. This ensures that the size of the collection of strings that we have to worry about of length up to n is $2^{n+1}-2$, and so the advice size we need is $\lceil \log_2((2^{n+1}-2)+2)\rceil = n+1$. So $\mathrm{A\text{-}NPMV}_\mathrm{t}$-sel indeed is in $\mathrm{NP}/_{\mathrm{strong}}\, n+1 \cap \mathrm{coNP}/_{\mathrm{strong}}\, n+1$, where by $/_{\mathrm{strong}}$ we mean the strong advice model. Keeping in mind that $(\mathrm{P}/1) - \mathrm{NPMV\text{-}sel} \neq \emptyset$, we state the discussion of this paragraph as follows.

PROPOSITION 5.11.

1. $\mathrm{A\text{-}P\text{-}sel} \subsetneq \mathrm{P}/_{\mathrm{strong}}\, n+1$.
2. $\mathrm{A\text{-}NP\text{-}sel} \subsetneq (\mathrm{NP} \cap \mathrm{coNP})/_{\mathrm{strong}}\, n+1$.
3. $\mathrm{A\text{-}NPMV}_\mathrm{t}\mathrm{\text{-}sel} \subsetneq \mathrm{NP}/_{\mathrm{strong}}\, n+1 \cap \mathrm{coNP}/_{\mathrm{strong}}\, n+1$.

It is natural to wonder whether there is some natural "NPSV" analogue of part 1 of Theorem 5.9 that yields an $(\mathrm{NP}\cap\mathrm{coNP})/n{+}1$ result. In light of part 2 of Theorem 5.8, one in particular might hope that $\mathrm{A}_\ell\mathrm{\text{-}NPSV\text{-}sel} \cap \mathrm{NP} \subseteq (\mathrm{NP}\cap\mathrm{coNP})/n+1$. However, we see no way of proving this. The key obstacle is obtaining the "$\mathrm{NP}\cap\mathrm{coNP}$" of "$(\mathrm{NP}\cap\mathrm{coNP})/n+1$" when dealing with partial function classes such as NPSV. One can patch one's way around the partialness of NPSV, namely, by using the advice to give not just an appropriate (via part 3 of Proposition 3.3) string from the set but also a certificate proving that the string is in the set. However, doing so requires enough bits to encode the certificate. So what one obtains, if one modifies part 1 of Theorem 5.9 carefully via the strategy just described, is the following result, in which NP_1 denotes the class, introduced by Kintala and Fischer in 1977 ([KF77]; see also, for contrast, [KF80]), of sets that are accepted by NP machines that use at most a linear number of nondeterministic moves.

THEOREM 5.12. $\mathrm{A}_\ell\mathrm{\text{-}NPSV\text{-}sel} \cap \mathrm{NP}_1 \subseteq (\mathrm{NP}\cap\mathrm{coNP})/\mathcal{O}(n)$.

The following is crucial in seeing that Theorem 5.12 holds. When given advice asserting that our A_ℓ-NPSV-selective set, B, contains y as a "magic" (in the sense of part 3 of Proposition 3.3) string for length $|y|$, only if the advice contains a correct certificate for $y \in B$ do we go forward and run our NPSV-selector function f, i.e., $f(x, y)$, where x, $|x| = |y|$, is the string whose membership we are interested in. So we run f only on cases in which at least one of the inputs on which we run f is clearly known by us to belong to B. By the definition of NPSV-selectivity, f on such inputs will always be defined. Thus, working hand in hand with the advice and the definition of NPSV-selectivity, we have in effect locally total-ized f, and this is what creates an $\mathrm{NP} \cap \mathrm{coNP}$ set and yields the $(\mathrm{NP} \cap \mathrm{coNP})/\mathcal{O}(n)$ result.

To the best of our knowledge, Theorem 5.12 is only the second time in the literature that the direct local totalization of NPSV functions appears. The first time was in Hemaspaandra et al. [HNOS96b], though in contrast there they were in the context of a Ko-type [Ko83] advice setting, i.e., one in which (ignoring certificate space) a linear number of linear-sized strings were being provided. Here, we have given a second application of their local totalization technique, this time in the context of the (ignoring certificate space) one-string advice of the sort provided by part 3 of Proposition 3.3.

Finally, we would like to remark on results involving weak associativity. If we assume the NPSV- or NPMV-selectors to be only weakly associative, we are merely able to show containment in $\mathrm{NP}/n{+}1$.

6. Printability and nonimmunity. Associativity yields additional simplicity properties. Let us consider nonimmunity results (i.e., presence of infinite subsets). For general NPSV-selective sets and thus also for NP- and P-selective sets, Theorem 6.1 holds. In contrast, for associatively NPSV-selective sets, we have Theorem 6.2. To see how these bounds relate, note that $\mathrm{UP}^{\mathrm{NP}} \subseteq \mathrm{NP}^{\mathrm{NP}}$. Of course, $\mathrm{NP}^{\mathrm{NP}}$ is synonymous with Σ_2^p; however, so as to ensure the relationship to $\mathrm{UP}^{\mathrm{NP}}$—and the way we'll use those "NP"s in our proofs—shows clearly, throughout this section we will write $\mathrm{NP}^{\mathrm{NP}}$ rather than Σ_2^p.

THEOREM 6.1. *Every infinite* NPSV*-selective set B has an infinite* $\mathrm{FP}^{B \oplus \mathrm{NP}^{\mathrm{NP}}}$*-printable subset.*

THEOREM 6.2. *Every infinite set B that is either* A-NPSV*-selective or* A_ℓ-NP*-selective has an infinite* $\mathrm{FP}^{B \oplus \mathrm{UP}^{\mathrm{NP}}}$*-printable subset.*

COROLLARY 6.3.

1. *Every infinite* P*-selective set B has an infinite* $\mathrm{FP}^{B \oplus \mathrm{NP}^{\mathrm{NP}}}$*-printable subset.*
2. *Every infinite* NP*-selective set B has an infinite* $\mathrm{FP}^{B \oplus \mathrm{NP}^{\mathrm{NP}}}$*-printable subset.*
3. *Every infinite* A-P*-selective (or even* A_ℓ-P*-selective) set B has an infinite* $\mathrm{FP}^{B \oplus \mathrm{UP}^{\mathrm{NP}}}$*-printable subset.*
4. *Every infinite* A-NP*-selective set B has an infinite* $\mathrm{FP}^{B \oplus \mathrm{UP}^{\mathrm{NP}}}$*-printable subset.*

Proof of Theorem 6.1. Let B be an infinite NPSV-selective set. By Fact 2.3 there exists a commutative NPSV-selector f for B. Let $n \in \mathbb{N}$. Consider the f-induced digraph on $B^{=n}$ (in the sense defined in section 3) and call it $G_{B,n,f}$. Since f is always defined when at least one of its inputs is in B, it follows that $G_{B,n,f}$ is an s-tournament. By applying a standard result (stated as Theorem 3.1 in [HO02]; the theorem refers to tournaments rather than s-tournaments, but the difference does not affect this result) of Ko [Ko83] about tournament theory to $G_{B,n,f}$, we have that for each $n \in \mathbb{N}$ there exists a set $D_n \subseteq B^{=n}$ such that

1. $\|D_n\| \leq n+1$, and
2. for every string $x \in B^{=n}$, there exists a string $y \in D_n$ such that $y \in \mathit{set}\text{-}f(x,y)$.

Of course, when $B^{=n} = \emptyset$, $G_{B,n,f}$ has no nodes and $D_n = \emptyset$.

Since f is a selector for B and thus $f(u,v) = v$ for all $u \notin B$ and all $v \in B$, we can state that for each $n \in \mathbb{N}$ such that $B^{=n} \neq \emptyset$ there exists a set $D_n \subseteq B^{=n}$ such that

1. $\|D_n\| \leq n+1$, and
2. for every string $x \in \Sigma^n$, there exists a string $y \in D_n$ such that $y \in \mathit{set}\text{-}f(x,y)$.

Let *setcode* denote some standard encoding (of finite sets into strings) that is computable and invertible in polynomial time. Let $\langle \cdot, \cdot \rangle$ be a standard polynomial-time computable, polynomial-time invertible pairing function. Define

$$E = \{\langle 1^n, \mathit{setcode}(\{y_1, y_2, \ldots, y_j\})\rangle \mid 1 \leq j \leq n+1 \wedge |y_1| = \cdots = |y_j| = n \wedge (\forall x \in \Sigma^n)(\exists i : 1 \leq i \leq j)[x \notin \mathit{set}\text{-}f(x, y_i)]\}.$$

Observe that the condition $x \notin \mathit{set}\text{-}f(x,y_i)$ is equivalent to $\mathit{set}\text{-}f(x,y_i) \in \{\emptyset, \{y_i\}\}$. Clearly, $E \in \mathrm{coNP}$. For all n, let $g(1^n)$ denote the lexicographically largest string ω such that $\langle 1^n, \omega\rangle \in E$ if such a string ω exists. (Note that in light of the discussion earlier in this proof and the properties and single-valuedness of f, it holds that, for each n such that $B^{=n} \neq \emptyset$, there exists at least one string w such that $\langle 1^n, w\rangle \in E$.) Otherwise $g(1^n)$ is undefined. Given 1^n one can by applying a prefix search procedure compute $g(1^n)$ in polynomial time with the help of oracle queries to the oracle $F = \{\langle 1^n, z\rangle \mid n \in \mathbb{N} \wedge z \in \Sigma^* \wedge (\exists \omega \in \Sigma^*)[\langle 1^n, z\omega\rangle \in E]\}$. Clearly, $F \in \mathrm{NP}^{\mathrm{NP}}$ and hence $g \in \mathrm{FP}^{\mathrm{NP}^{\mathrm{NP}}}$.

We claim that, for all n, if $B^{=n} \neq \emptyset$, then for every $\langle 1^n, \mathit{setcode}(\{y_1, y_2, \ldots, y_j\})\rangle \in E$ $(1 \leq j \leq n+1)$—and so in particular for $\langle 1^n, g(1^n)\rangle$—there exists an index i, $1 \leq i \leq j$, such that $y_i \in B^{=n}$. To see this, suppose that $B^{=n} \neq \emptyset$, yet there exists some $\langle 1^n, \mathit{setcode}(\{y_1, y_2, \ldots, y_j\})\rangle \in E$ $(1 \leq j \leq n+1)$ such that, for all i, $1 \leq i \leq j$, $y_i \notin B^{=n}$. Let $x \in B^{=n}$. Due to the definition of E we have that there exists an index i, $1 \leq i \leq j$, such that either $\mathit{set}\text{-}f(x,y_i) = \{y_i\}$ or $f(x,y_i)$ is undefined, both of which are impossible since f is an NPSV-selector for B and so, when exactly one of its arguments (x in this case) is in B, it outputs that argument.

Consider the set

$$C = B \cap \bigcup_{\{i \in \mathbb{N} \mid g(1^i) \text{ is defined}\}} \mathit{setcode}^{-1}(g(1^i)).$$

Note that C is infinite if B is infinite. Also $C \subseteq B$. In order to see that C is $\mathrm{FP}^{B \oplus \mathrm{NP}^{\mathrm{NP}}}$-printable, consider the following DPTM M: On input 1^n, $M(1^n)$ computes $g(1^0), g(1^1), \ldots, g(1^n)$ with the help of the $\mathrm{NP}^{\mathrm{NP}}$ part, F, of its $B \oplus \mathrm{NP}^{\mathrm{NP}}$ oracle. It then computes the polynomial-sized set $\bigcup_{\{i \leq n \mid g(1^i) \text{ is defined}\}} \mathit{setcode}^{-1}(g(1^i))$ and then, using the B part of its oracle, finds which of these strings are in B and outputs them. □

Theorem 6.1 should be contrasted with the result of Hemaspaandra et al. [HOZZ04] that P-sel is not (weak-$\mathrm{FP}^{\mathrm{NP}^{\mathrm{NP}}}$-rankable)-immune.

Proof of Theorem 6.2. In light of part 3 of Corollary 4.7, it actually holds that A_ℓ-NP-sel $\supseteq$ A-NP-sel = A-NPSV-sel. Hence it suffices to show the claim for A_ℓ-NP-selective sets. Let B be an infinite A_ℓ-NP-selective set. By part 2 of Theorem 4.2,

A_ℓ-NP-sel = A_ℓC-NP-sel. So, let f be a commutative $NPSV_t$-selector for B that is associative at each length.

Consider the function *score* that is defined as

$$score(x) = \|\{z \in \Sigma^{|x|} \mid f(x,z) = x\}\|.$$

Note that for each $n \in \mathbb{N}$ and for each $x \in \Sigma^n$, (a) $x \notin B \implies score(x) \leq 2^n - \|B^{=n}\|$ and (b) $x \in B \implies score(x) \geq 2^n - \|B^{=n}\| + 1$. So for all $x \in B$ and all $y \notin B$ such that $|x| = |y|$, $score(y) < score(x)$. It follows from the fact that f is commutative, total, single-valued, and length-associative that for all n, $\max\{score(y) \mid |y| = n\} = 2^n$ (consider Proposition 3.3 applied to the f-induced digraph on the nodes $(\Sigma^*)^{=n}$). So for each n there exists exactly one string at length n, call it d_n, such that $score(d_n) = \max\{score(y) \mid |y| = n\} = 2^n$. Thus, the set

$$F = \{\langle 1^n, z\rangle \mid n \in \mathbb{N} \wedge z \in (\Sigma^*)^{\leq n} \wedge (\exists \omega : |\omega| = n - |z|)[score(z\omega) = 2^n]\}$$

is in UP^{NP}. (To see this, note that $score(x) = 2^{|x|}$ is in our setting equivalent to $(\forall y \in \Sigma^{|x|})[x = y \vee f(x,y) \neq y]$.) Given 1^n, computing d_n can be done in polynomial time with the help of oracle F. Let $C = B \cap \{d_i \mid i \geq 0\}$. Clearly, since B is infinite, by our choice of d_i we have that C is both infinite and $FP^{B \oplus UP^{NP}}$-printable. □

So, Theorems 6.1 and 6.2 have been proven.[6]

It follows from the proof of Theorem 6.2 that any associatively NPSV selective set for which there is an infinite set of lengths at which we know it is not empty is in fact not coNP-immune. This is a *very* weak type of partial census information.

Definition 6.4. *We say a set B is hintable if there is an infinite tally set, $T \subseteq 1^*$, such that $T \in P$ and*

$$(\forall i)[1^i \in T \implies \|B^{=i}\| \neq 0].$$

Theorem 6.5. *Any hintable* A-NPSV-*selective (or even* A_ℓ-NPSV-*selective) set has an infinite* coNP *subset (i.e., languages in* A_ℓ-NPSV-sel$\cap$Hintable *are not* coNP-*immune).*

Proof. Let B be an infinite hintable A_ℓ-NPSV-selective set. Let the tally set T be a hint set for B in the sense of Definition 6.4. Let f be a commutative NPSV-selector for B that is total at each length n such that $B^{=n} \neq \emptyset$ and associative at each length n such that $B^{=n} \neq \emptyset$. Such a selector for B exists by Corollary 4.7. Define the function *score* as in the proof of Theorem 6.2. Since B is A_ℓ-NPSV-selective, we have that, for all n, if $B^{=n} \neq \emptyset$, then there exists a unique string of length n having a score of 2^n and it is in B. (If $B^{=n} = \emptyset$, then there might be no string at length n having a score of 2^n.) The set

$$C = \{x \mid 1^{|x|} \in T \wedge score(x) = 2^n\}$$

is an infinite subset of B and is in coNP since it will hold that $C = \{x \mid 1^{|x|} \in T \wedge (\forall y : |y| = |x|)[y = x \vee y \notin set\text{-}f(x,y)]\}$. □

[6] If one cares about the number of queries to B needed in the printability claims of Theorems 6.1 and 6.2 and Corollary 6.3, the following, which was communicated to us (regarding a somewhat weaker set of results in an earlier version of this paper) by Till Tantau [Tan01], can be observed: One can limit one's queries to B to a logarithmic number by putting the queries to B into a Toda-like ordering ([Tod91]; see also [HT03]) and binary searching to find which are in and which are out. (For the nondeterministic selectivity claims, one will use the power of NP in the oracle to Toda-order the strings. Also, the partial-function cases do not present a problem, e.g., on those one can modify for the partial case the "bubblesort"-type proof given by Hemaspaandra and Torenvliet [HT03, Proof of Lemma 4.5].

7. Are all P-selective sets associatively P-selective? Many of our results give simplicity properties of A-P-sel. It is natural to wonder whether in fact A-P-sel is all of P-sel, that is, whether all P-selective sets are associatively P-selective. In this section, we prove upper bounds on the power needed to associatively select sets. One consequence, Corollary 7.3, is that if P = NP, then A-P-sel and P-sel are equal.

THEOREM 7.1.

1. *Every* $\mathrm{NPMV_t}$*-selective set has a single-valued, commutative selector function in* $\mathrm{FP_t}^{\mathrm{NP}}$ *that is associative.*
2. *Every* NPMV*-selective set* A *has a single-valued, commutative selector function in* $\mathrm{FP_t}^{\mathrm{NP}}$ *that is associative on* A*.*
3. *Every* NPMV*-selective set has a single-valued, commutative selector function in* $\mathrm{FP_t}^{\Sigma_2^p}$ *that is associative.*

Proof. We first prove part 1 of the result. Let A be an $\mathrm{NPMV_t}$-selective set. Let f be an $\mathrm{NPMV_t}$-selector for A. Without loss of generality, assume f to be commutative (see Fact 2.3). For every pair of strings x and y, ω is called a connector of x and y if and only if either (a) $\omega \in \mathit{set}\text{-}f(x,\omega)$ and $y \in \mathit{set}\text{-}f(\omega,y)$, or (b) $x \in \mathit{set}\text{-}f(x,\omega)$ and $\omega \in \mathit{set}\text{-}f(\omega,y)$. Note that for any two strings x and y, there always exist connectors of x and y, namely, x and y. Define the function g by setting, for all x and y,

$$g(x,y) = \begin{cases} y & \text{if the lexicographically smallest connector } \omega \text{ of } x \text{ and } y \text{ yields } \omega \in \mathit{set}\text{-}f(x,\omega) \wedge y \in \mathit{set}\text{-}f(\omega,y) \text{ but not } x \in \mathit{set}\text{-}f(x,\omega) \wedge \omega \in \mathit{set}\text{-}f(\omega,y), \\ x & \text{if the lexicographically smallest connector } \omega \text{ of } x \text{ and } y \text{ yields } x \in \mathit{set}\text{-}f(x,\omega) \wedge \omega \in \mathit{set}\text{-}f(\omega,y) \text{ but not } \omega \in \mathit{set}\text{-}f(x,\omega) \wedge y \in \mathit{set}\text{-}f(\omega,y), \\ \max(x,y) & \text{if the lexicographically smallest connector } \omega \text{ of } x \text{ and } y \text{ yields } \omega \in \mathit{set}\text{-}f(x,\omega) \wedge y \in \mathit{set}\text{-}f(\omega,y) \text{ and } x \in \mathit{set}\text{-}f(x,\omega) \wedge \omega \in \mathit{set}\text{-}f(\omega,y). \end{cases}$$

Note that g is a single-valued function in $\mathrm{FP_t}^{\mathrm{NP}}$. To compute $g(x,y)$ the NP oracle is first used to—via either binary search or prefix search—compute the lexicographically smallest connector of x and y, and then, via two more queries, is used to determine which of the three cases from the definition of g is applicable. Note that g is commutative and self-contained.

To see that g is even a selector for A, suppose that there are strings x and y such that $x \in A$ and $y \notin A$ yet $g(x,y) = y$. Let ω be the lexicographically smallest connector for x and y. It follows that either (a) $\omega \in \mathit{set}\text{-}f(x,\omega) \wedge y \in \mathit{set}\text{-}f(\omega,y)$ but not $x \in \mathit{set}\text{-}f(x,\omega) \wedge \omega \in \mathit{set}\text{-}f(\omega,y)$, or (b) $y = \max(x,y)$, $\omega \in \mathit{set}\text{-}f(x,\omega) \wedge y \in \mathit{set}\text{-}f(\omega,y)$, and $x \in \mathit{set}\text{-}f(x,\omega) \wedge \omega \in \mathit{set}\text{-}f(\omega,y)$. However, in both cases the properties of a selector function (in this case f) imply that if $x \in A$, so are ω and y, contradicting our assumption.

It remains to show that g is associative. Suppose that g is not associative. Let a,b,c be a counterexample to the associativity of g. Since g is a single-valued, commutative self-contained function, it follows from Corollary 3.4 that both $||\{a,b,c\}|| = 3$ and the g-induced digraph on a,b,c is a 3-cycle (plus three self-loops). Without loss of generality, suppose that $g(a,b) = b$, $g(b,c) = c$, and $g(a,c) = a$. Let ω be the lexicographically smallest connector among all connectors for a and b, for b and c, and for a and c.

Suppose that ω is a connector for a and b. Due to symmetry we can make that assumption without loss of generality. Since $g(a,b) = b$ there are two possible scenarios.

$$h(x,y) = \begin{cases} y & \text{if } x \text{ and } y \text{ have a connector } z \text{ such that } z \leq_{\text{lex}} \max(x,y) \text{ and the lexicographically smallest connector } \omega \text{ of } x \text{ and } y \text{ yields } \omega \in \textit{set-f}(x,\omega) \wedge y \in \textit{set-f}(\omega,y) \text{ but not } x \in \textit{set-f}(x,\omega) \wedge \omega \in \textit{set-f}(\omega,y), \\ x & \text{if } x \text{ and } y \text{ have a connector } z \text{ such that } z \leq_{\text{lex}} \max(x,y) \text{ and the lexicographically smallest connector } \omega \text{ of } x \text{ and } y \text{ yields } x \in \textit{set-f}(x,\omega) \wedge \omega \in \textit{set-f}(\omega,y) \text{ but not } \omega \in \textit{set-f}(x,\omega) \wedge y \in \textit{set-f}(\omega,y), \\ \max(x,y) & \text{if } x \text{ and } y \text{ have a connector } z \text{ such that } z \leq_{\text{lex}} \max(x,y) \text{ and the lexicographically smallest connector } \omega \text{ of } x \text{ and } y \text{ yields both } \omega \in \textit{set-f}(x,\omega) \wedge y \in \textit{set-f}(\omega,y) \text{ and } x \in \textit{set-f}(x,\omega) \wedge \omega \in \textit{set-f}(\omega,y), \\ \max(x,y) & \text{if } x \text{ and } y \text{ have no connector } z \text{ such that } z \leq_{\text{lex}} \max(x,y). \end{cases}$$

FIG. 7.1. *Definition of the function h used in the proof of Theorem* 7.1.

Case 1 ($\omega \in$ *set-f*$(a,\omega) \wedge b \in$ *set-f*(ω,b) but not $a \in$ *set-f*$(a,\omega) \wedge \omega \in$ *set-f*(ω,b)):
Since f is total, *set-f*$(\omega,c) \neq \emptyset$. If *set-f*$(\omega,c) = \{c\}$, then ω is also a connector for a and c contradicting, in light of the definition of g and the fact that in this case $\omega \in$ *set-f*(a,ω) and *set-f*$(\omega,c) = \{c\}$ yet $\omega \notin$ *set-f*(ω,c), $g(a,c) = a$. If *set-f*$(\omega,c) = \{\omega\}$, then ω is also a connector for b and c in such a way as to similarly contradict $g(b,c) = c$. If *set-f*$(\omega,c) = \{\omega,c\}$, there are two possibilities. The first is that *set-f*$(\omega,b) = \{b\}$. However, that implies $g(b,c) = b$, which contradicts our assumption that $g(b,c) = c$. The second possibility is that *set-f*$(\omega,b) = \{b,\omega\}$. This implies (under the rules of Case 1 and the subcase we are in) that *set-f*$(a,\omega) = \{\omega\}$. However, this implies that $g(c,a) = c$, which contradicts our assumption that $g(a,c) = a$.

Case 2 ($b = \max(a,b)$, $\omega \in$ *set-f*$(a,\omega) \wedge b \in$ *set-f*(ω,b), and $a \in$ *set-f*$(a,\omega) \wedge \omega \in$ *set-f*(ω,b)):
Since f is total, $f(\omega,c)$ is defined. If *set-f*$(\omega,c) = \{c\}$, then we have $g(a,c) = c$, contradicting our assumption that $g(a,c) = a$. If *set-f*$(\omega,c) = \{\omega\}$, then we have $g(b,c) = b$, contradicting our assumption that $g(b,c) = c$. If *set-f*$(\omega,c) = \{\omega,c\}$, then the definition of g and the assumed values of g imply that $b = \max(a,b)$, $a = \max(a,c)$, and $c = \max(b,c)$, which contradicts $||\{a,b,c\}|| = 3$.

It follows that g is associative, which completes the proof of part 1.

We next show part 2. This can be shown quite similarly, and so we will sketch only the main differences to the proof of part 1. Let A be an NPMV-selective set. Let f be an NPMV-selector for A. Without loss of generality, assume f to be commutative (Fact 2.3). Observe that $f(x,y)$ and also that one of $f(x,x)$ and $f(y,y)$ is defined whenever at least one of x and y is in A. Thus, whenever at least one of x and y is in A there exists a connector z of x and y such that $z \leq_{\text{lex}} \max(x,y)$. Define the function h, for all x and y, as shown in Figure 7.1. Note that h is a single-valued $\text{FP}_{\text{t}}^{\text{NP}}$ function that is commutative. Similarly to the proof of part 1, one can now show that h is a selector function for A that is associative on A. For the latter note that any connector z of two strings $x, y \in A$ is also in A.

Part 3 follows from the proofs of parts 1 and 2. The function h as defined in the proof of part 2 is a total, commutative, and single-valued FP^{NP} function. We can

now repeat the entire proof of part 1 but replacing f by h and NPMV$_t$ by $\mathrm{FP}_t^{\mathrm{NP}}$ in that proof. It follows that the function g as defined in the proof of part 1 is now in our modified version a total single-valued $\mathrm{FP}^{\Sigma_2^p}$-selector for A that is commutative and associative. □

COROLLARY 7.2.

1. *Every* P*-selective set, and even every* FP*-selective set, has a commutative selector function in* $\mathrm{FP}_t^{\mathrm{NP}}$ *that is associative.*
2. *Every* NP*-selective set has a commutative selector function in* $\mathrm{FP}_t^{\mathrm{NP}}$ *that is associative.*
3. *Every* NPSV*-selective set A has a commutative selector function in* $\mathrm{FP}_t^{\mathrm{NP}}$ *that is associative on A.*
4. *Every* NPSV*-selective set has a commutative selector function in* $\mathrm{FP}_t^{\Sigma_2^p}$ *that is associative.*

The above corollary is a straightforward consequence of Theorem 7.1. Regarding the "every FP-selective" in part 1, recall FP-sel = P-sel from Fact 2.3.

COROLLARY 7.3. *If* P = NP*, then* P-sel = A-P-sel*,* FP-sel = A-FP-sel*,* NP-sel = A-NP-sel*,* NPSV-sel = A-NPSV-sel*,* NPMV$_t$-sel = A-NPMV$_t$-sel*, and* NPMV-sel = A-NPMV-sel*.*

The above corollary follows directly from Theorem 7.1 and and Corollary 7.2. Regarding the NPSV-sel = A-NPSV-sel and NPMV-sel = A-NPMV-sel conclusions of the above corollary, recall that $\mathrm{P} = \mathrm{NP} \iff \mathrm{P} = \Sigma_2^p$.

Theorem 5.3 provides a sufficient condition, based on an algebraic property for selector functions, for P-sel $\subseteq \mathrm{P}/\mathcal{O}(n)$. If one were interested only in *structural complexity-class-collapse* sufficient conditions, Theorem 5.3 would be no improvement over the P = NP sufficient condition implicit in the result of Hemaspaandra and Torenvliet that P-sel $\subseteq \mathrm{NP}/\mathcal{O}(n)$ (and thus P-sel $\subseteq \mathrm{P}/\mathcal{O}(n)$ if P = NP), since in light of Corollary 7.3 the best known structural complexity-class-collapse condition sufficient to imply P-sel = A-P-sel is also the collapse P = NP. However, we feel that this is the wrong view, and that "P-sel = A-P-sel" is probably a fundamentally different type of assumption than P = NP. For example, if P-sel = A-P-sel were in fact equivalent to P = NP, then P = NP would (trivially) not just be a sufficient condition for P-sel = A-P-sel but would also be a necessary condition. In fact, not only is P = NP not known to be necessary for P-sel = A-P-sel, but in fact no structural complexity-class-collapse condition—not even very weak collapses like P = UP or P = ZPP—is known to be necessary. Our point here is that, though by Corollary 7.3 P = NP is one way to achieve P-sel = A-P-sel, we conjecture that it is unlikely that one can prove that it characterizes P-sel = A-P-sel. And thus our P-sel = A-P-sel sufficient condition for P-sel $\subseteq \mathrm{P}/\mathcal{O}(n)$ is best viewed as a new algebraic sufficient condition quite different from the known (and extremely demanding) structural complexity-class-collapse sufficient conditions.

We do have a structural complexity-class-collapse condition, namely, the collapse of the polynomial hierarchy, that follows from the assumption that every NPMV-selective set is in fact associatively NPMV-selective. (S_2 is the symmetric alternation class of Canetti [Can96] and Russell and Sundaram [RS98]. Note that a collapse to $\mathrm{S}_2^{\mathrm{NP}}$ is known to be at least as strong as a collapse to $\mathrm{ZPP}^{\Sigma_2^p}$ or to Σ_3^p, since $\mathrm{S}_2^{\mathrm{NP}} \subseteq \mathrm{ZPP}^{\Sigma_2^p} \subseteq \Sigma_3^p$ ([CCHO03]; see that paper also for the definition of $\mathrm{S}_2^{\mathrm{NP}}$).)

THEOREM 7.4. NPMV-sel = A-NPMV-sel $\implies$ PH = $\mathrm{S}_2^{\mathrm{NP}}$.

Proof. Assume that NPMV-sel = A-NPMV-sel. By Corollary 5.10 we have that A-NPMV-sel $\subseteq \mathrm{NP}/n{+}1 \cap \mathrm{coNP}/n{+}1$. So under our assumption, NPMV-sel $\subseteq$

$\mathrm{NP}/n{+}1 \cap \mathrm{coNP}/n{+}1$. However, it is well known that $\mathrm{NP} \subseteq$ NPMV-sel. Putting all the pieces together we obtain $\mathrm{NP} \subseteq \mathrm{coNP}/n{+}1$. This itself, by the recent strengthening of Yap's theorem [Yap83] by Cai et al. [CCHO03], implies that $\mathrm{PH} = \mathrm{S}_2^{\mathrm{NP}}$ (equivalently, $\mathrm{PH} = \mathrm{NP}^{\mathrm{NP}^{\mathrm{NP}}} = \mathrm{ZPP}^{\mathrm{NP}^{\mathrm{NP}}} = \mathrm{S}_2^{\mathrm{NP}}$). □

Finally, we mention two open issues. First, can one prove a complete or partial converse of other parts of Corollary 7.3? Second, one would ultimately like to know whether all P-selective sets have linear deterministic advice, i.e., whether P-sel $\subseteq \mathrm{P}/\mathcal{O}(n)$. This paper gives a new sufficient condition for that, namely, P-sel $\subseteq \mathrm{P}/\mathcal{O}(n)$ if all P-selective sets have associative (or even merely length-associative) selector functions.

Acknowledgments. We thank the COCOON 2001 and SICOMP referees for their helpful comments, and we thank Dieter Kratsch, Haiko Müller, Till Tantau, and Leen Torenvliet for helpful conversations and comments. We are deeply grateful to Klaus Wagner for his valuable suggestions.

REFERENCES

[AH92] E. Allender and L. Hemachandra, *Lower bounds for the low hierarchy*, J. ACM, 39 (1992), pp. 234–251.

[BJG00] J. Bang-Jensen and G. Gutin, *Digraphs*, Springer-Verlag, London, 2001.

[BLS84] R. V. Book, T. J. Long, and A. L. Selman, *Quantitative relativizations of complexity classes*, SIAM J. Comput., 13 (1984), pp. 461–487.

[BLS85] R. Book, T. Long, and A. Selman, *Qualitative relativizations of complexity classes*, J. Comput. System Sci., 30 (1985), pp. 395–413.

[BS92] J. Balcázar and U. Schöning, *Logarithmic advice classes*, Theoret. Comput. Sci., 99 (1992), pp. 279–290.

[Cai01] J. Cai, $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2001, pp. 620–629.

[Cam94] P. Cameron, *Combinatorics: Topics, Techniques, Algorithms*, Cambridge University Press, Cambridge, UK, 1994.

[Can96] R. Canetti, *More on BPP and the polynomial-time hierarchy*, Inform. Process. Lett., 57 (1996), pp. 237–241.

[CCHO03] J. Cai, V. Chakaravarthy, L. Hemaspaandra, and M. Ogihara, *Some Karp–Lipton-type theorems based on* S_2, in Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2607, Springer-Verlag, New York, 2003, pp. 535–546.

[HHN+95] L. Hemaspaandra, A. Hoene, A. Naik, M. Ogiwara, A. Selman, T. Thierauf, and J. Wang, *Nondeterministically selective sets*, Internat. J. Found. Comput. Sci., 6 (1995), pp. 403–416.

[HNOS96a] E. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman, *P-selective sets and reducing search to decision vs. self-reducibility*, J. Comput. System Sci., 53 (1996), pp. 194–209.

[HNOS96b] L. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman, *Computing solutions uniquely collapses the polynomial hierarchy*, SIAM J. Comput., 25 (1996), pp. 697–708.

[HNP98] L. Hemaspaandra, C. Nasipak, and K. Parkins, *A note on linear-nondeterminism, linear-sized, Karp–Lipton advice for the P-selective sets*, J. Univ. Comput. Sci., 4 (1998), pp. 670–674.

[HO02] L. Hemaspaandra and M. Ogihara, *The Complexity Theory Companion*, Springer-Verlag, Berlin, 2002.

[HOW02] L. Hemaspaandra, M. Ogihara, and G. Wechsung, *Reducing the number of solutions of NP functions*, J. Comput. System Sci., 64 (2002), pp. 311–328.

[HOZZ04] L. Hemaspaandra, M. Ogihara, M. Zaki, and M. Zimand, *The complexity of finding top-Toda-equivalence-class members*, in Proceedings of the 6th Latin American Symposium on Theoretical Informatics, Lecture Notes in Comput. Sci., Springer-Verlag, 2004, to appear.

[HR99] L. Hemaspaandra and J. Rothe, *Creating strong, total, commutative, associative*

one-way functions from any one-way function in complexity theory, J. Comput. System Sci., 58 (1999), pp. 648–659.

[HT96] L. Hemaspaandra and L. Torenvliet, *Optimal advice*, Theoret. Comput. Sci., 154 (1996), pp. 367–377.

[HT03] L. Hemaspaandra and L. Torenvliet, *Theory of Semi-Feasible Algorithms*, Springer-Verlag, Berlin, 2003.

[HY84] J. Hartmanis and Y. Yesha, *Computation times of NP sets of different densities*, Theoret. Comput. Sci., 34 (1984), pp. 17–32.

[KF77] C. Kintala and P. Fischer, *Computations with a restricted number of nondeterministic steps*, in Proceedings of the 9th ACM Symposium on Theory of Computing, ACM Press, New York, 1977, pp. 178–185.

[KF80] C. M. R. Kintala and P. Fischer, *Refining nondeterminism in relativized polynomial-time bounded computations*, SIAM J. Comput., 9 (1980), pp. 46–53.

[KL80] R. Karp and R. Lipton, *Some connections between nonuniform and uniform complexity classes*, in Proceedings of the 12th ACM Symposium on Theory of Computing, ACM Press, New York, 1980, pp. 302–309; an extended version has also appeared as *Turing machines that take advice*, Enseign. Math. (2), 28 (1982), pp. 191–209.

[Kle52] S. Kleene, *Introduction to Metamathematics*, D. van Nostrand, New York, 1952.

[Ko83] K. Ko, *On self-reducibility and weak P-selectivity*, J. Comput. System Sci., 26 (1983), pp. 209–221.

[Ko87] K. Ko, *On helping by robust oracle machines*, Theoret. Comput. Sci., 52 (1987), pp. 15–36.

[KS85] K.-I Ko and U. Schöning, *On circuit-size complexity and the low hierarchy in NP*, SIAM J. Comput., 14 (1985), pp. 41–51.

[Moo68] J. Moon, *Topics on Tournaments*, Holt, Rinehart, and Winston, New York, 1968.

[MS72] A. Meyer and L. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential space*, in Proceedings of the 13th IEEE Symposium on Switching and Automata Theory, IEEE, Piscataway, NJ, 1972, pp. 125–129.

[NRRS98] A. Naik, J. Rogers, J. Royer, and A. Selman, *A hierarchy based on output multiplicity*, Theoret. Comput. Sci., 207 (1998), pp. 131–157.

[NS99] A. Naik and A. Selman, *Adaptive versus nonadaptive queries to NP and P-selective sets*, Comput. Complexity, 8 (1999), pp. 169–187.

[Ogi96] M. Ogihara, *Functions computable with limited access to NP*, Inform. Process. Lett., 58 (1996), pp. 35–38.

[RS97] M. Rabi and A. Sherman, *An observation on associative one-way functions in complexity theory*, Inform. Process. Lett., 64 (1997), pp. 239–244.

[RS98] A. Russell and R. Sundaram, *Symmetric alternation captures BPP*, Comput. Complexity, 7 (1998), pp. 152–162.

[Sch83] U. Schöning, *A low and a high hierarchy within NP*, J. Comput. System Sci., 27 (1983), pp. 14–28.

[Sel79] A. Selman, *P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP*, Math. Systems Theory, 13 (1979), pp. 55–65.

[Sel81] A. Selman, *Some observations on NP real numbers and P-selective sets*, J. Comput. System Sci., 23 (1981), pp. 326–332.

[Sel82a] A. Selman, *Analogues of semirecursive sets and effective reducibilities to the study of NP complexity*, Inform. and Control, 52 (1982), pp. 36–51.

[Sel82b] A. Selman, *Reductions on NP and P-selective sets*, Theoret. Comput. Sci., 19 (1982), pp. 287–304.

[Sel96] A. Selman, *Much ado about functions*, in Proceedings of the 11th Annual IEEE Conference on Computational Complexity, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 198–212.

[Siv99] D. Sivakumar, *On membership comparable sets*, J. Comput. System Sci., 59 (1999), pp. 270–280.

[Sto76] L. Stockmeyer, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 1–22.

[Tan01] T. Tantau, *personal communication*, 2001.

[Tha03] M. Thakur, *On Optimal Advice for P-Selective Sets*, Technical Report TR-819, Department of Computer Science, University of Rochester, Rochester, NY, 2003.

[Tod91] S. Toda, *On polynomial-time truth-table reducibilities of intractable sets to P-selective sets*, Math. Systems Theory, 24 (1991), pp. 69–82.

[Yap83] C. Yap, *Some consequences of non-uniform conditions on uniform classes*, Theoret. Comput. Sci., 26 (1983), pp. 287–300.

SIAM J. COMPUT.
Vol. 33, No. 6, pp. 1338–1368

GRAPHS WITH TINY VECTOR CHROMATIC NUMBERS AND HUGE CHROMATIC NUMBERS*

URIEL FEIGE†, MICHAEL LANGBERG‡, AND GIDEON SCHECHTMAN§

Abstract. Karger, Motwani, and Sudan [*J. ACM*, 45 (1998), pp. 246–265] introduced the notion of a vector coloring of a graph. In particular, they showed that every k-colorable graph is also vector k-colorable, and that for constant k, graphs that are vector k-colorable can be colored by roughly $\Delta^{1-2/k}$ colors. Here Δ is the maximum degree in the graph and is assumed to be of the order of n^{δ} for some $0 < \delta < 1$. Their results play a major role in the best approximation algorithms used for coloring and for maximum independent sets.

We show that for every positive integer k there are graphs that are vector k-colorable but do not have independent sets significantly larger than $n/\Delta^{1-2/k}$ (and hence cannot be colored with significantly fewer than $\Delta^{1-2/k}$ colors). For $k = O(\log n/\log\log n)$ we show vector k-colorable graphs that do not have independent sets of size $(\log n)^c$, for some constant c. This shows that the vector chromatic number does not approximate the chromatic number within factors better than $n/\text{polylog}n$.

As part of our proof, we analyze "property testing" algorithms that distinguish between graphs that have an independent set of size n/k, and graphs that are "far" from having such an independent set. Our bounds on the sample size improve previous bounds of Goldreich, Goldwasser, and Ron [*J. ACM*, 45 (1998), pp. 653–750] for this problem.

Key words. semidefinite programming, chromatic number, independent set, approximation algorithms, property testing

AMS subject classifications. 68R05, 05C15, 90C22

DOI. 10.1137/S0097539703431391

1. Introduction.

An independent set in a graph G is a set of vertices that induce a subgraph which does not contain any edges. The size of the maximum independent set in G is denoted by $\alpha(G)$. For an integer k, a k-coloring of G is a function $\sigma : V \to [1 \dots k]$ which assigns colors to the vertices of G. A valid k-coloring of G is a coloring in which each color class is an independent set. The chromatic number $\chi(G)$ of G is the smallest k for which there exists a valid k-coloring of G.

Finding $\alpha(G)$ and $\chi(G)$ are fundamental NP-hard problems, closely related by the inequality $\alpha(G)\chi(G) \geq n$. Given G, the question of estimating the value of $\alpha(G)$ ($\chi(G)$) or finding large independent sets (small colorings) in G has been studied extensively. Let G be a graph of size n. Both $\chi(G)$ and $\alpha(G)$ can be approximated within a ratio of $O(\frac{n(\log\log n)^2}{\log^3 n})$ (see [Hal93, Fei02]). It is known that unless NP = ZPP, neither $\alpha(G)$ nor $\chi(G)$ can be approximated within a ratio of $n^{1-\varepsilon}$ for any $\varepsilon > 0$ [Hås99, FK98]. Under stronger complexity assumptions, there is some $0 < \delta < 1$ such that neither problem can be approximated within a ratio of $n/2^{\log^{\delta} n}$ [Kho01].

*Received by the editors July 9, 2003; accepted for publication (in revised form) March 2, 2004; published electronically August 6, 2004.

http://www.siam.org/journals/sicomp/33-6/43139.html

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel 76100 (uriel.feige@weizmann.ac.il). This author was supported in part by the Israel Science Foundation (grant 236/02).

‡Department of Computer Science, California Institute of Technology, Pasadena, CA 91125 (mikel@cs.caltech.edu). This author's work was done while studying at the Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel 76100.

§Department of Mathematics, Weizmann Institute of Science, Rehovot, Israel 76100 (gideon.schechtman@weizmann.ac.il). The work of this author was supported in part by the Israel Science Foundation (grant 154/01).

The approximation ratios for these problems significantly improve in graphs that have very large independent sets or very small chromatic numbers. The algorithms achieving the best ratios in these cases [KMS98, AK98, BK97, HNZ01] are all based on the idea of *vector coloring*, introduced by Karger, Motwani, and Sudan [KMS98].

DEFINITION 1.1. *A vector k-coloring of a graph is an assignment of unit vectors to its vertices such that, for every edge, the inner product of the vectors assigned to its endpoints is at most (in the sense that it can only be more negative) $-1/(k-1)$.*

Every k-colorable graph is also vector k-colorable (by identifying each color class with one vertex of a perfect $(k-1)$-dimensional simplex centered at the origin). Moreover, unlike the chromatic number, a vector k-coloring (when it exists) can be found in polynomial time using semidefinite programming (up to arbitrarily small error in the inner products). Given a vector k-coloring of a graph, Karger, Motwani, and Sudan show how to color a graph with roughly $\Delta^{1-2/k}$ colors, where Δ is the maximum degree in the graph. (In comparison, the technique of inductive coloring might use $\Delta+1$ colors.) In fact, they show how to find an independent set of size roughly $n/\Delta^{1-2/k}$. Combined with other ideas, this leads to coloring algorithms and algorithms for finding independent sets with the best currently known performance guarantees. For example, there is a polynomial time algorithm that colors 3-colorable graphs with roughly $n^{3/14}$ colors [BK97]. For nonconstant values of k, it is known how to find an independent set of size $\Omega(\log n)$ in a vector $\log n$-colorable graph.

There have also been negative results regarding vector k-colorable graphs. Examples appearing in [KMS98] (and improved by Alon and Szegedy) show vector 3-colorable graphs that do not have independent sets larger than roughly $n^{0.95}$. The case of nonconstant k is addressed in [Fei97] (technically, the results there deal with the Lovász theta function, which is even a stronger notion than vector coloring), where graphs that are vector $2^{O(\sqrt{\log n})}$-colorable are shown not to have independent sets larger than $2^{O(\sqrt{\log n})}$. In all these negative examples, the vertex sets of the graphs involved can be viewed as a subset of $\{0,1\}^n$, with two vertices connected by an edge if their Hamming distance is larger than some prespecified value.

Our results. In this work we present a different family of graphs with stronger negative properties. For every constant $k>2$ and every $\varepsilon>0$, we show graphs that are vector k-colorable, with $\alpha(G) \le n/\Delta^{1-\frac{2}{k}-\varepsilon}$. This essentially matches the positive results of [KMS98]. As a function of n rather than Δ, we show vector 3-colorable graphs with $\alpha(G) < n^{0.843}$. Moreover, for $k = O(\log n/\log\log n)$, we show vector k-colorable graphs with $\alpha(G) \le (\log n)^c$, for some universal c. This shows that the vector coloring number by itself does not approximate the chromatic number within a ratio better than $n/\text{polylog}n$. Another consequence of this (that is touched upon in Remark 2.1 in section 2) is that certain semidefinite programs do not approximate the size of the maximum independent set with a ratio better than $n/\text{polylog}n$.

THEOREM 1.2.

1. *For every constant $\varepsilon>0$ and constant $k>2$, there are infinitely many graphs G that are vector k-colorable and satisfy $\alpha(G) \le n/\Delta^{1-\frac{2}{k}-\varepsilon}$, where n is the number of vertices in G, Δ is the maximum degree in G, and $\Delta > n^{\delta}$ for some constant $\delta>0$.*
2. *For some constant c, there are infinitely many graphs G that are vector $O(\frac{\log n}{\log\log n})$-colorable and satisfy $\alpha(G) \le (\log n)^c$.*
3. *There are infinitely many graphs G that are vector 3-colorable and satisfy $\alpha(G) \le n^{0.843}$.*

Observe that if Theorem 1.2(1) is proven for some graph with n vertices and

maximum degree Δ, then for every positive integer c it also holds for some graph with $c \cdot n$ vertices and the same maximum degree Δ. (Simply make c disjoint copies of the original graph.) Hence the theorem becomes stronger as Δ becomes larger in comparison to n. We guarantee that Δ can be taken at least as large as n^{δ} for some δ that depends on ϵ and k. For every fixed ϵ and k, the value of δ can be taken as a fixed constant bounded away from 0 and independent of n. We have not made an attempt to find the tightest possible relation between δ and ϵ and k.

Proof techniques. The graphs that we use are essentially the same graphs that were used in [FS02] to show integrality gaps for semidefinite programs for Max-Cut. Namely, they are obtained by placing n points at random on a d-dimensional unit sphere and connecting two points by an edge if the inner product of their respective vectors is below $-1/(k-1)$. Such graphs are necessarily vector k-colorable, as the embedding on the sphere is a vector k-coloring. So the bulk of the work is in proving that they have no large independent set. For this we use a two-phase plan. First we consider a continuous (infinite) graph, where every point on the sphere is a vertex and two vertices are connected by an edge if the inner product of their respective vectors is below $-1/(k-1)$. On this continuous graph we use certain symmetrization techniques in order to analyze its properties. Specifically, we prove certain inequalities regarding its expansion. In the second phase, we consider a (finite) random vertex induced subgraph of the continuous graph. Based on the expansion properties of the continuous graph, we show that the random sample has no large independent set.

Two remarks are in order here. One is that it is very important for our bounds that the final random graph does not contain too many vertices compared to the dimension d. A small number of vertices implies low degree, and allows for a more favorable relation between the maximum degree and the chromatic number. For this reason we cannot use the continuous graph (or a finite discrete approximation of the continuous graph) as is. Its degree is very large compared to its chromatic number. We thus consider the graph obtained by randomly sampling the vertices of the continuous graph. In section 7, we follow a suggestion of Luca Trevisan (private communication) and present an alternative proof of Theorem 1.2(1) by considering the graph obtained by randomly sampling the *edges* of the continuous graph. It appears that parts (2) and (3) of Theorem 1.2 cannot be proven using edge sampling.

The other remark is that we do not get an explicit graph as our example, but rather a random graph (or a distribution on graphs). This is to some extent unavoidable, given that there are no known efficient deterministic constructions of Ramsey graphs (graphs in which the size of the maximum independent set and maximum clique are both bounded by some polylog in n). The graphs we construct (when $k = \log n / \log\log n$) are Ramsey graphs, because it can be shown that the maximum clique size is never larger than the vector coloring number.

Property testing. The following problem in property testing is addressed by Goldreich, Goldwasser, and Ron [GGR98]. For some value of $\rho < 1$, consider a graph with the following "promise": either it has an independent set of size ρn, or it is far from any such graph, in the sense that any vertex-induced subgraph of ρn vertices induces at least εn^2 edges. One wants an algorithm that samples as few vertices as possible, looks at the subgraph induced on them, and, based on the size of the maximum independent set in that subgraph, decides correctly (with high probability) which of the two cases above hold. In [GGR98] it is shown that a sample of size proportional to ε^{-4} suffices. We are in a somewhat similar situation when we move from the continuous graph to our random sample. The continuous graph is far from having an indepen-

dent set of *measure* ρ.[1] We want to take a sample as small as possible (its size will be denoted by s) such that the induced subgraph does not contain an independent set of size ρs. Luckily in our case, we can use a stronger guarantee on the continuous graph. We know that even every set of measure $\rho/2$ induces at least the measure of ε edges.[2] In this case we show that a sample size s proportional to $1/\varepsilon$ suffices. This dramatic improvement over the [GGR98] bound is crucial to the success of our analysis. We note that this improvement is not only based on our stronger guarantee on the continuous graph G. Even in exactly the same setting of [GGR98], we show that a sample size s proportional to ε^{-3} suffices. These results are of interest in the context of property testing regardless of their applications to the vector coloring issue.

Strict vector coloring. One may strengthen the notion of vector k-coloring by requiring that, for every edge, the inner product of the unit vectors corresponding to its endpoints be exactly $-1/(k-1)$, rather than at most $-1/(k-1)$. This is called a strict vector coloring. This notion is known to be equivalent to the theta function of Lovász [Lov79, KMS98]. Every k-colorable graph is also strictly vector k-colorable. As strict vector coloring is a stronger requirement than vector coloring, then it is possible that strict vector k-colorable graphs have smaller chromatic numbers than vector k-colorable graphs. So far, there are not any algorithmic techniques that can use this observation to further improve the approximation ratios for chromatic number or for independent set. We remark, however, that the negative results in this work apply only to vector coloring and not to strict vector coloring. It is an open question whether similar negative results are true for strict vector coloring, or equivalently, whether the same gaps (such as $n/\text{polylog}n$) can be shown between the value of the theta function and the size of the maximum independent set. Note that the weaker negative results of [Fei97] and some of the negative results in [KMS98] do apply also to strict vector coloring.

The remainder of this paper is organized as follows. In section 2 we briefly review the semidefinite programs that compute the vector chromatic number and its variants. In section 3 we present the continuous graph and analyze its properties. In section 4 we prove Theorem 1.2. Our results on property testing are presented in sections 5 and 6. In section 7 we present an alternative proof of Theorem 1.2(1). In section 8 we discuss some problems that remain open. Finally, in section 9 we present the proof of several technical lemmas that appear throughout our work.

2. The vector chromatic number and its variants. There are many equivalent ways to define the vector chromatic number and its variants. We will follow the definitions suggested in [KMS98, Cha02]. Let $G = (V, E)$ be a graph of size n. For convenience we will assume that $V = [1, \ldots, n]$. The semidefinite relaxations below assign unit vectors to every vertex $i \in V$. These unit vectors are to satisfy certain constraints which will in turn determine the value of the relaxations.

$COL_1(G)$ Minimize k
subject to
$\langle v_i, v_j \rangle \leq -\frac{1}{k-1}$ $\forall (i,j) \in E$,
$\langle v_i, v_i \rangle = 1$ $\forall i \in V$;

$COL_2(G)$ Minimize k

[1] As the continuous graph is infinite, terms such as the *number* of vertices are replaced by the continuous analog *measure*. The measure we use on the unit sphere is the natural measure, which associates with each subset its relative area on the sphere.

[2] The measure we use on the edge set is the product measure induced by the standard uniform measure on S^{d-1}.

subject to

$$\langle v_i, v_j \rangle = -\tfrac{1}{k-1} \qquad \forall (i,j) \in E,$$
$$\langle v_i, v_i \rangle = 1 \qquad \forall i \in V;$$

$COL_3(G)$ Minimize k

subject to

$$\langle v_i, v_j \rangle = -\tfrac{1}{k-1} \qquad \forall (i,j) \in E,$$
$$\langle v_i, v_j \rangle \geq -\tfrac{1}{k-1} \qquad \forall i,j \in V,$$
$$\langle v_i, v_i \rangle = 1 \qquad \forall i \in V.$$

The function $COL_1(G)$ is the vector chromatic number of G as defined in [KMS98]. The function $COL_2(G)$ is the strict vector chromatic number of G and is equal to the Lovász θ function on $\bar{G}$ [Lov79, KMS98], where $\bar{G}$ is the *complement* graph of G. Finally, the function $COL_3(G)$ will be referred to as the *strong* vector chromatic number as defined in [Sze94, Cha02]. Let $\omega(G)$ denote the size of the maximum clique in G; in the following we show that

$$\omega(G) \leq COL_1(G) \leq COL_2(G) \leq COL_3(G) \leq \chi(G).$$

It is not hard to verify that $COL_1(G) \leq COL_2(G) \leq COL_3(G)$. To show the other inequalities we need the following fact. For every integer k, the k unit vectors $\{v_1, \ldots, v_k\}$ that minimize the value of $\max_{i \neq j \in [1,\ldots,k]} \langle v_i, v_j \rangle$ are the vertices of a simplex in R^{k-1} centered at the origin. For each $i, j \in [1, \ldots, k]$, these vectors satisfy $\langle v_i, v_j \rangle = -\frac{1}{k-1}$.

Now to prove the inequality $COL_3(G) \leq \chi(G)$, consider a k-coloring σ of G. The coloring σ partitions the vertex set V into k color classes $\{V_1, \ldots, V_k\}$. Assigning each color class V_i with the corresponding vector v_i above, we obtain a valid assignment for $COL_3(G)$. To show that $\omega(G) \leq COL_1(G)$, consider a graph G with maximum clique size $\omega(G)$. To obtain a valid assignment of vectors to $COL_1(G)$ of value k we require that all pairs of vectors corresponding to the vertices of the maximum clique will have inner product of value at most $-\frac{1}{k-1}$. As mentioned above, this can happen only if $\omega(G) \leq k$.

Remark 2.1. The results of our work show a large gap between $COL_1(G)$ and $\chi(G)$ (Theorem 1.2). Combining these results with certain proof techniques appearing in [Sze94], a similar gap between $\omega(G)$ and $COL_3(G)$ can also be obtained. Details are omitted.

3. The continuous graph. Let d be a large constant, and let $S^{d-1} = \{v \in R^d \mid \|v\| = 1\}$ be the d-dimensional unit sphere. Let $G_k = (V, E)$ be the continuous graph in which (a) the vertex set V consists of all the points on the unit sphere S^{d-1}, and (b) the edge set E of G_k consists of all pairs of vertices whose respective vectors (from the origin) form an angle of at least $\arccos(-1/(k-1))$. As the size of V and E is infinite (and uncountable), terms such as the *number* of vertices in V will be replaced by the continuous analogue *measure*.

In this section we analyze various properties of G_k. Specifically, we show that G_k has certain expansion properties. We then use this fact in section 4 to prove the main theorem of our work. In our analysis, we will assume that the dimension d is (at least) a very large constant (our proofs rely on such d). Additional constants that will be presented in the remainder of this section are to be viewed as independent of d.

DEFINITION 3.1 (sphere measure). *Let μ be the normalized $(d-1)$-dimensional natural measure on S^{d-1}, and let μ^2 be the induced measure on $S^{d-1} \times S^{d-1}$. For any*

two (not necessarily disjoint) subsets A and B of V, we define the measure of edges from A to B as

$$E(A,B) = \mu^2\left(\{(x,y) \mid x \in A,\ y \in B,\ (x,y) \in E\}\right).$$

DEFINITION 3.2 (sphere caps). *Let $a \in [0,1]$ and $x \in S^{d-1}$. An a-cap centered at x is defined to be the set $\mathcal{C}_a = \{u \in S^{d-1} \mid \langle u,x\rangle \geq a\}$. Denote the measure of an a-cap by $\rho(a)$.*

A few remarks are appropriate. Notice that for every $x,\ x'$, an a-cap centered at x has the same measure as an a-cap centered at x' (sphere symmetry). Furthermore, notice that *large* caps have *small* corresponding values of a and vice versa. The value of $\rho(a) = \mu(\mathcal{C}_a)$ is approximately given by the following lemma proven, for example, in [FS02].

LEMMA 3.3. *Let $\mathcal{C}_a$ be an a-cap centered at some $x \in S^{d-1}$. There exists a constant $c > 0$ (independent of a and d) s.t. (such that)*

$$\frac{c}{\sqrt{d}}\left(1-a^2\right)^{\frac{d-1}{2}} \leq \mu(\mathcal{C}_a) = \rho(a) \leq \frac{1}{2}\left(1-a^2\right)^{\frac{d-1}{2}}.$$

LEMMA 3.4. *Let $\rho(\frac{1}{k-1})$ be the measure of a $\frac{1}{(k-1)}$-cap. Every vertex v in the graph G_k has degree $\rho(\frac{1}{k-1})$.*

Proof. Each vertex v in G_k is adjacent to every vertex in the $\frac{1}{(k-1)}$-cap centered at $-v$ (here $-v$ is the vertex antipodal to v). Hence, the measure of vertices adjacent to a given vertex v in G_k is that of a $\frac{1}{(k-1)}$-cap. ∎

The main property of G_k of interest to us is the measure of edges between any two given subsets of V of a specified size. We first prove that the sets in G_k which share the least amount of edge are caps with the same center.

THEOREM 3.5. *Let $1 \geq a > 0$, and let A and B be two (not necessarily disjoint) measurable sets in V of measure $\rho(a)$. Let x be an arbitrary vertex of S^{d-1}. The minimum of $E(A,B)$ is obtained when $A = B = \mathcal{C}_a$, where $\mathcal{C}_a$ is an a-cap of measure $\rho(a)$ centered at x.*

The proof of Theorem 3.5 is based on symmetrization techniques similar to those presented in [BT76, FS02]. We prove Theorem 3.5 in section 9. We now turn to analyzing the measure of edges between caps of measure $\rho(a)$. Namely, we study the value of $E(\mathcal{C}_a,\mathcal{C}_a)$. For *large* values of a, it is not hard to see that $\mathcal{C}_a$ will not induce any edges. This follows from the fact that any two vertices u and v in $\mathcal{C}_a$ satisfy $\langle u,v\rangle \geq 2a^2-1$. Hence, in the following we consider values of a which satisfy $2a^2-1 < -\frac{1}{k-1}$. For such values of a we show that $E(\mathcal{C}_a,\mathcal{C}_a)$ is proportional to $\mu^2(E)\rho(a)^{2+\frac{2}{k-2}}$ (which we denote by $\lambda(a)$). Notice that two random subsets A and B of G_k of measure $\rho(a)$ are expected to satisfy $E(A,B) = \mu^2(E)\rho(a)^2$, which is fairly close to $\lambda(a)$. Actually, we show that $E(\mathcal{C}_a,\mathcal{C}_a)$ is in the range $[(1-c\sqrt{\log(d)/d})^{\frac{d-1}{2}}\lambda(a),\lambda(a)]$ for some large constant c. It is not hard to verify that the term $(1-c\sqrt{\log(d)/d})^{\frac{d-1}{2}}$ is negligible compared to $\lambda(a)$, once d is taken to be large enough.

THEOREM 3.6. *Let $\sqrt{\frac{k-2}{2(k-1)}} > a > 0$ and $k > 2$ be constant. Let $x \in S^{d-1}$, and let $\mathcal{C}_a$ be an a-cap centered at x. Let $\varepsilon(a)$ be the value of $E(\mathcal{C}_a,\mathcal{C}_a)$. Finally let*

$$\lambda(a) = \left(\left(1-\frac{1}{(k-1)^2}\right)\left(1-\frac{2(k-1)}{k-2}a^2\right)\right)^{\frac{d-1}{2}}.$$

Then $\varepsilon(a) \in [(1-c\sqrt{\log(d)/d})^{\frac{d-1}{2}}\lambda(a),\lambda(a)]$ for some constant $c > 0$.

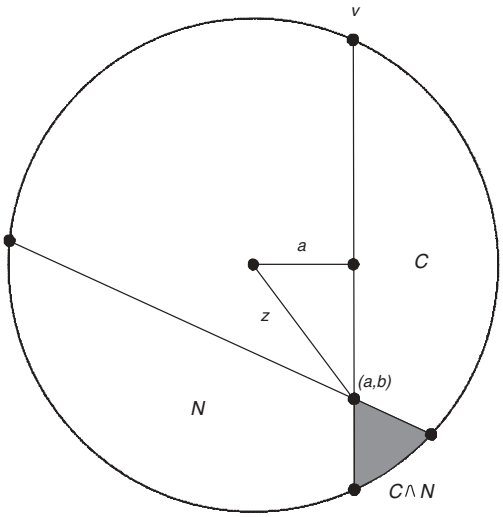


FIG. 1. *Projecting S^{d-1} onto the two-dimensional subspace $\{(r_1, r_2, 0, \ldots, 0) \mid r_1, r_2 \in R\}$ of R^d, we obtain the circle above. C is the projection of an a-cap $\mathcal{C}_a$ centered at $(1, 0, \ldots, 0)$ (where a is the distance of the cap from the origin). The vertex $v = (a, \sqrt{1-a^2}, 0, \ldots, 0) \in \mathcal{C}_a$ is on the boundary of $\mathcal{C}_a$. N is the projection on the sphere of the set of points $N(v)$ that are adjacent to v (i.e., form an angle of at least* $\arccos(-1/(k-1))$ *with v). The shaded section is $C \cap N$ (the projection of $\mathcal{C}_a \cap N(v)$). The point (a, b) is the closest point of the projection of $\mathcal{C}_a \cap N(v)$ to the origin. It is not hard to verify that $b^2 = (1/(k-1) + a^2)^2/(1-a^2)$. Finally, we denote the value of $\sqrt{a^2+b^2}$ by z. Claim 3.7 addresses the measure of $\mathcal{C}_a \cap N(v)$ and states that it is essentially the measure of a z-cap. This is done by studying the points in S^{d-1} whose projection falls* close *to (a, b). Roughly speaking, we first show that such points are in $\mathcal{C}_a \cap N(v)$; then, using Claim 9.7, we show that the measure of these points is essentially the measure of a z-cap.*

Proof. Let $x \in S^{d-1}$. Let $\mathcal{C}_a$ be an a-cap centered at x. W.l.o.g. we will assume that $x = (1, 0, \ldots, 0)$. Consider a vertex $v \in \mathcal{C}_a$ on the boundary of $\mathcal{C}_a$. Let $N(v)$ be the set of vertices adjacent to v. We start by computing the measure of vertices that are neighbors of v and are in the cap $\mathcal{C}_a$, i.e., the measure of $N(v) \cap \mathcal{C}_a = \{u = (u_1, \ldots, u_d) \in S^{d-1} \mid u_1 \geq a \text{ and } \langle v, u \rangle \leq -1/(k-1)\}$.

CLAIM 3.7. *Let a, k be as in Theorem 3.6. Let $v = (a, \sqrt{1-a^2}, 0, \ldots, 0)$ be a vertex on the boundary of $\mathcal{C}_a$.Let $N(v)$ be the set of neighbors of v. Let $z = \sqrt{a^2 + \frac{(1/(k-1)+a^2)^2}{1-a^2}}$.Finally let $\delta = c\sqrt{\frac{\log(d)}{d}}$ for a sufficiently large constant c. The measure of vertices in $N(v) \cap \mathcal{C}_a$ satisfies*

$$(1-\delta)^{\frac{d-1}{2}} \left(1-z^2\right)^{\frac{d-1}{2}} \leq \mu(N(v) \cap \mathcal{C}_a) \leq \left(1-z^2\right)^{\frac{d-1}{2}}.$$

Claim 3.7 addresses the measure of $\mathcal{C}_a \cap N(v)$, and states that it is essentially the measure of a z-cap $\left(\text{where } z = \sqrt{a^2 + \frac{(1/(k-1)+a^2)^2}{1-a^2}}\right)$. To prove Claim 3.7 we study the measure of certain restricted sets in S^{d-1}. These sets are studied in Claim 9.7 of section 9. Claims 3.7 and 9.7 are depicted in Figure 1 and proven in section 9.

To complete the proof of Theorem 3.6, let a, z, δ be as in Claim 3.7. For a vertex $v \in \mathcal{C}_a$ let $N(v) \cap \mathcal{C}_a$ be the set of vertices adjacent to v in $\mathcal{C}_a$. For the upper bound, notice that of all vertices in $\mathcal{C}_a$, the vertices v in which $\mu(N(v) \cap \mathcal{C}_a)$ is largest are the vertices on the boundary of $\mathcal{C}_a$. By Claim 3.7, for these vertices $\mu(N(v) \cap \mathcal{C}_a)$ is bounded by $(1-z^2)^{\frac{d-1}{2}}$. We thus conclude that $\varepsilon(a)$ is bounded by the measure of

vertices in $\mathcal{C}_a$ times $(1-z^2)^{\frac{d-1}{2}}$. That is,

$$\varepsilon(a) \le \rho(a)(1-z^2)^{\frac{d-1}{2}} \le (1-a^2)\left(1-\left(a^2+\frac{(1/(k-1)+a^2)^2}{1-a^2}\right)\right)^{\frac{d-1}{2}} = \lambda(a).$$

As for the lower bound, let $w=(w_1,w_2,\ldots,w_d)$ be a vertex in $\mathcal{C}_a$ with first coordinate w_1 of value $a+\delta$. Consider any vertex $v=(v_1,v_2,\ldots,v_d)\in\mathcal{C}_a$ with first coordinate v_1 of value less than $a+\delta$. It is not hard to verify that the measure of $N(v)\cap\mathcal{C}_a$ is greater than the measure of $N(w)\cap\mathcal{C}_a$. Using an analysis similar to that of Claim 3.7, we have that $\mu(N(w)\cap\mathcal{C}_a)$ is greater than or equal to

$$(1-c\delta)^{\frac{d-1}{2}}\left(1-\left(a^2+\frac{(1/(k-1)+a(a+\delta))^2}{1-(a+\delta)^2}\right)\right)^{\frac{d-1}{2}} \ge (1-c\delta)^{\frac{d-1}{2}}\left(1-z^2\right)^{\frac{d-1}{2}}$$

for a sufficiently large constant c (which changes values between both sides of the second inequality). Furthermore, for our choice of δ, the measure of vertices $v=(v_1,v_2,\ldots,v_d)$ in $\mathcal{C}_a$ with $v_1\le a+\delta$ is at least $\rho(a)/2$ (Claim 9.3). Hence, we conclude that $E(\mathcal{C}_a,\mathcal{C}_a)$ is at least $\frac{\rho(a)}{2}(1-c\delta)^{\frac{d-1}{2}}\left(1-z^2\right)^{\frac{d-1}{2}}$. Simplifying the above expression, we conclude our assertion. □

Theorem 3.6 addresses the case in which k is constant and the caps considered are both of measure $\rho(a)$ for a constant value of a. For the proof of Theorem 1.2(2) we also need to address nonconstant values of a and k which depend on d.

THEOREM 3.8. *Let $a=(\log(d)/d)^{\frac{1}{4}}$. Let k satisfy $1/(k-1)=a^2$. Let $x\in S^{d-1}$, and let $\mathcal{C}_a$ be an a-cap centered at x. Let $\varepsilon(a)$ be the value of $E(\mathcal{C}_a,\mathcal{C}_a)$. The value of $\varepsilon(a)$ is in the range*

$$\left[\frac{1}{poly(d)}\left(1-\frac{2(k-1)}{k-2}a^2\right)^{\frac{d-1}{2}},\ \left(1-\frac{2(k-1)}{k-2}a^2\right)^{\frac{d-1}{2}}\right].$$

The outline of the proof of Theorem 3.8 is similar to that of Theorem 3.6. A full proof appears in section 9.

DEFINITION 3.9. *Let $\rho<1$. A graph $G=(V,E)$ is said to be pairwise $\langle\rho,\varepsilon\rangle$-connected iff every two (not necessarily disjoint) subsets A and B of V of measure ρ satisfy $E(A,B)\ge\varepsilon$.*

Combining Theorems 3.5, 3.6, and 3.8 we obtain the following result.

COROLLARY 3.10. *Let a,k, $\varepsilon(a)$ be defined as in Theorem* 3.6 *or* 3.8. *The graph G_k is pairwise $\langle\rho(a),\varepsilon(a)\rangle$-connected.*

Roughly speaking, Corollary 3.10 addresses the expansion properties of the continuous graph G_k. In section 5, we show that these properties imply certain upper bounds on the independence number of a small random sample of G_k. Namely, we prove the following theorem.

THEOREM 3.11. *Let a,k, $\varepsilon(a)$ be defined as in Theorem* 3.6 *or* 3.8. *Let H be a random sample of s vertices of G_k (according to the uniform distribution on S^{d-1}). Let c be a sufficiently large constant. If $s\ge c\frac{\rho(a)}{\varepsilon(a)}\log^2(1/\rho(a))$, then the probability that $\alpha(H)>e^2\rho(a)s$ is at most* 1/4.

In the following section, Theorem 3.11 is used to prove the main result of this work, Theorem 1.2. The proof of Theorem 3.11 will be presented in section 5.2.

4. Proof of Theorem 1.2. Recall that we are looking for a graph H for which both the vector chromatic number and the size of the maximum independent set are

small. The graphs H that we present are random subgraphs of the graphs G_k defined in section 3. The three assertions of Theorem 1.2 are all proven similarly; the main difference among their proofs is the choice of parameters used. To avoid confusion, we restate Theorem 1.2 using a slightly different notation then that appearing in the original presentation.

THEOREM 1.2 (restated).

1. *For every constant $\gamma > 0$ and constant $k > 2$, there are infinitely many graphs H that are vector k-colorable and satisfy $\alpha(H) \le s/\Delta_H^{1-\frac{2}{k}-\gamma}$, where s is the number of vertices in H, Δ_H is the maximum degree in H, and $\Delta_H > s^\delta$ for some constant $\delta > 0$.*
2. *For some constant c, there are infinitely many graphs H of size s that are $O(\frac{\log s}{\log\log s})$ vector colorable and satisfy $\alpha(H) \le (\log s)^c$.*
3. *There are infinitely many graphs H of size s which are vector 3-colorable and satisfy $\alpha(H) \le s^{0.843}$.*

Proof of Theorem 1.2(1). Let $k > 2$ be constant. Let $\gamma > 0$ be an arbitrarily small constant. Let c be a sufficiently large constant, and let $a = \gamma/c$. Let $G = G_k = (V, E)$ be the continuous graph from section 3. (Here and in the remainder of the proof, we assume that the dimension d of the graph G_k is taken to be significantly larger than $1/\gamma$.) Finally, let $\Delta = \rho(\frac{1}{k-1})$ be the measure of vertices adjacent to any given vertex of G.

Recall (from Corollary 3.10) that G is pairwise $\langle \rho(a), \varepsilon(a)\rangle$-connected. Let $\rho = \rho(a)$ and $\varepsilon = \varepsilon(a)$. This implies (Theorem 3.11) that with probability $\ge 3/4$ a random subset H of G of size $s \ge c\frac{\rho}{\varepsilon}\log^2(1/\rho)$ satisfies $\alpha(H) \le e^2\rho s$. (Recall that c is a sufficiently large constant.) We start by simplifying the expression bounding s.

CLAIM 4.1. *A random subset H of G of size $s = 1/(\Delta\rho^{\frac{k}{k-2}+\gamma})$ satisfies $\alpha(H) \le e^2\rho s = e^2/(\Delta\rho^{\frac{2}{k-2}+\gamma})$ with probability $\ge 3/4$.*

Proof. It suffices to prove that $s = 1/(\Delta\rho^{\frac{k}{k-2}+\gamma}) \ge c\frac{\rho}{\varepsilon}\log^2(1/\rho)$. The claim follows (by basic calculations) from the fact that ε can be bounded by $\Delta\rho^{\frac{2(k-1)}{k-2}+\gamma}$. By Theorem 3.6, we have that

$$\varepsilon(a) = \left[\left(1 - c\left(\sqrt{\frac{\log(d)}{d}}\right)\right)\left(1 - \frac{1}{(k-1)^2}\right)\left(1 - \frac{2(k-1)}{k-2}a^2\right)\right]^{\frac{d-1}{2}}.$$

It is not hard to verify that $(1-c(\sqrt{\log(d)/d}))^{\frac{d-1}{2}} > \rho^{\gamma/2}$. Furthermore, by Lemma 3.3 we have that $(1 - \frac{1}{(k-1)^2})^{\frac{d-1}{2}} > \Delta$. Hence, $\varepsilon(a) \ge \rho^{\gamma/2}(1 - \frac{2(k-1)}{k-2}a^2)^{\frac{d-1}{2}}\Delta$. Recall that a was defined as γ/c for a sufficiently large constant c. This implies that

$$\gamma \ge 2\frac{a^2\left(\frac{2(k-1)}{k-2}\right)^2}{1 - a^2\frac{2(k-1)}{k-2}}$$

($k > 2$ is constant). This expression is designed to fit the requirement appearing in Claim 9.1 (of section 9). Now by Claim 9.1, it holds that $(1 - \frac{2(k-1)}{k-2}a^2)^{\frac{d-1}{2}} \ge \rho^{\frac{2(k-1)}{k-2}+\frac{\gamma}{2}}$. We conclude that $\varepsilon(a) \ge \Delta\rho^{\frac{2(k-1)}{k-2}+\gamma}$. ∎

Let H be a random subgraph of G of size $s = 1/(\Delta\rho^{\frac{k}{k-2}+\gamma})$. We will show that H satisfies the asserted conditions with probability greater than 1/2. First notice that any subgraph of G is vector k-colorable, including the subgraph H. Second, by

Claim 4.1, $\alpha(H) \leq e^2/(\Delta\rho^{\frac{2}{k-2}+\gamma})$ with probability $\geq 3/4$. It is left to analyze the maximum degree of H.

CLAIM 4.2. *With probability greater than* $3/4$*, the maximum degree* Δ_H *of the subgraph* H *is in the range* $[\frac{1}{2}\Delta s, 2\Delta s]$.

Proof. Consider a vertex $h \in H$. Let d_h be the degree of h. As every vertex in G is of degree Δ, the expected value of d_h is $\Delta(s-1)$. Thus (using standard bounds) the probability that d_h deviates from its expectation by more than a constant fraction of its expectation is at most $2^{-\Omega(\Delta s)}$. The probability that some vertex in H has degree $\notin [\frac{1}{2}\Delta s, 2\Delta s]$ is thus at most $2^{\log(s)-\Omega(\Delta s)} \leq 3/4$ for our choice of s. □

The first assertion of Theorem 1.2 now follows using basic calculations. □

Proof of Theorem 1.2(2). Let d be a large constant. Let $k-1 = \sqrt{d/\log(d)}$. Let $a^2 = \frac{1}{k-1}$. Let $G = G_k = (V, E)$ be the continuous graph from section 3.

Recall (Corollary 3.10) that G is pairwise $\langle \rho(a), \varepsilon(a)\rangle$-connected. Let $\rho = \rho(a)$ and $\varepsilon = \varepsilon(a)$. This implies (Theorem 3.11) that with probability $\geq 3/4$ a random subset H of size $s \geq c\frac{\rho}{\varepsilon}\log^2(1/\rho)$ satisfies $\alpha(H) \leq e^2\rho s$ (here c is a sufficiently large constant). As before we start by simplifying the expression bounding s.

CLAIM 4.3. *There exists a constant* γ *s.t. with probability at least* $3/4$ *a random set* H *of* G *of size* $s = \frac{d^\gamma}{\rho}\log^2(1/\rho)$ *satisfies* $\alpha(H) \leq e^2\rho s$ *with probability* $\geq 3/4$.

Proof. Recall that $k-1 = \sqrt{d/\log(d)}$, $a^2 = \frac{1}{k-1}$. As before, it suffices to bound $\varepsilon = \varepsilon(a)$. By Theorem 3.8, we have

$$\varepsilon \geq \frac{1}{poly(d)}\left(1 - a^2\frac{2(k-1)}{k-2}\right)^{\frac{d-1}{2}}.$$

Furthermore, using Claims 9.1 and 9.2 (of section 9), we obtain

$$\left(1 - a^2\frac{2(k-1)}{k-2}\right)^{\frac{d-1}{2}} = \left(1 - a^2\left(2 + \frac{2}{k-2}\right)\right)^{\frac{d-1}{2}} \geq \frac{1}{poly(d)}\left(1-a^2\right)^{d-1} \geq \frac{\rho^2}{poly(d)}.$$

We conclude that there exists a constant γ such that $\varepsilon \geq \frac{\rho^2}{d^\gamma}$. □

Let H be a random subset of vertices of G of size $s = \frac{d^\gamma}{\rho}\log^2(1/\rho)$. Notice that $\log(s) = \theta(\sqrt{d\log d})$ and $s \leq \theta(\frac{d^{\gamma+2}}{\rho})$. By definition, G is k vector colorable. This implies that any subgraph of G (including that induced by H) is $k = O\big(\frac{\log(s)}{\log(\log(s))}\big)$ vector colorable, which completes the proof of the first part of our assertion. For the second part of our assertion, by Claim 4.3 the subset H does not have an independent set of size $e^2\rho s \leq \log^\gamma(s)$ with probability at least $3/4$ (for some different constant γ). □

Proof of Theorem 1.2(3). The proof follows the line of proof appearing above. In general, we use the graph $G = G_3$, but this time the value of a is set to be $a = 0.36$. Again, G is pairwise $\langle \rho(a), \varepsilon(a)\rangle$-connected, where $\varepsilon(a)$ can be bounded by approximately $(\frac{3}{4}(1-4a^2))^{\frac{d-1}{2}}$. Let $\rho = \rho(a)$ and $\varepsilon = \varepsilon(a)$. By Theorem 3.11, a random subset H of size $s \geq c\frac{\rho}{\varepsilon}\log^2(1/\rho)$ does not have an independent set of size $e^2\rho s$ (with probability $\geq 3/4$). Computing the value of $\log_s \rho s$, we obtain our assertion. We would like to note that results of a similar nature can be obtained using the above techniques for any value of k. □

5. Random sampling. We now turn to proving Theorem 3.11 stated in section 3. This is done in two steps. In section 5.1 we prove results analogous to those presented in Theorem 3.11 when the graphs considered are finite. In section 5.2 we

show that our analysis extends to the continuous case (of section 3) as well. Finally, in section 5.3, we continue the study of finite graphs, and obtain results of independent interest in the context of property testing.

Let G be a graph of size n which does not have an independent set of size ρn (i.e., $\alpha(G) < \rho n$). Let H be a random subgraph of G of size s (i.e., H is the subgraph induced by a random subset of vertices in G of size s). In this section we study the minimal value of s for which $\alpha(H) \leq \rho s$ with high probability.

In general, if our only assumption on G is that $\alpha(G) < \rho n$, we cannot hope to set s to be smaller than n. Hence, we strengthen our assumption on G, to graphs G which not only satisfy $\alpha(G) \leq \rho n$ but are also *far* from having an independent set of size ρn. (We defer defining the exact notion of "far" until later in this discussion.) That is, given a graph G which is *far* from having an independent set of size ρn, we ask for the minimal value of s for which (with high probability) a random subgraph of size s does not have an independent set of size ρs. This question (and many other closely related ones) have been studied in [GGR98] under the title of *property testing*.

In [GGR98], a graph G of size n is said to be ε-far from having an independent set of size ρn if any set of size ρn in G has at least εn^2 induced edges. It was shown in [GGR98] that if G is ε-far from having an independent set of size ρn, then with high probability a random subgraph of size $s = \frac{c \log(1/\varepsilon)\rho}{\varepsilon^4}$, for a sufficiently large constant c, does not have an independent set of size ρs.

The results of [GGR98] do not suffice for the proof (as we present it) of Theorem 1.2. We thus turn to strengthening their results. To do so, we introduce a stronger notion of being "ε-far." Roughly speaking, we prove that, under our new notion of distance, choosing s to be of size $\frac{\rho}{\varepsilon}$ suffices. Furthermore, by applying our proof techniques on the original notion of distance presented in [GGR98], we improve the result of [GGR98] stated above and obtain a sample size proportional to $1/\varepsilon^3$. In section 6 we continue to study the original notion of ε-far from [GGR98] and present a lower bound on the sample size which is proportional to $1/\varepsilon^2$. The proof techniques used in this section are based on the techniques appearing in [GGR98] and [AK02]. (In the latter, property testing of the chromatic number is considered.) We start with the following definitions (which are finite versions of those given in section 3.)

DEFINITION 5.1. *Let A and B be (not necessarily disjoint) subsets of G. For each vertex $v \in A$ let $d_v(B)$ be the number of neighbors v has in B. Let $E(A,B) = \sum_{v \in A} d_v(B)$.*

DEFINITION 5.2. *Let $\rho < 1$. A graph $G = (V,E)$ is said to be $\langle \rho, \varepsilon \rangle$-connected iff every subset A of V of size ρn satisfies $E(A,A) \geq \varepsilon n^2$ (i.e., the number of edges in the subgraph induced by A is greater than $\frac{\varepsilon}{2} n^2$).*

Notice that $\varepsilon \leq \rho^2$. Furthermore, notice that a graph G is $\langle \rho, \varepsilon \rangle$-connected iff G is $\varepsilon/2$-far (by the definitions presented in [GGR98]) from having an independent set of size ρn.

DEFINITION 5.3. *Let $\rho < 1$. A graph $G = (V,E)$ is said to be* pairwise $\langle \rho, \varepsilon \rangle$*-connected iff every two (not necessarily disjoint) subsets A and B of V of size ρn satisfy $E(A,B) \geq \varepsilon n^2$.*

As mentioned above, for $\langle \rho, \varepsilon \rangle$-connected and pairwise $\langle \rho, \varepsilon \rangle$-connected graphs G, we study the minimal value of s for which a random subgraph H of G of size s satisfies $\alpha(H) \leq \rho s$ with high probability. Namely, we analyze the probability that a random subset H of G satisfies $\alpha(H) \leq \rho s$ (as a function of ρ, ε, and the sample size s). The main idea behind our proof is as follows. Given a sample size s, we start by bounding the probability that a random subset R of G of size $k > \rho s$ is an independent set.

Then, using the standard union bound on all subsets R of H of size greater than ρs, we bound the probability that $\alpha(H) > \rho s$.

Throughout this section we analyze the properties of random subsets H which are assumed to be *small*. Namely, we assume that the value of s and the parameters ρ, ε, and n satisfy (a) $s < c\sqrt{n}$ and (b) $s < c\rho n$ for a sufficiently small constant c. In our applications (and also in standard ones) these assumptions hold.

In section 5.1 we analyze the above proof strategy and show that it suffices to bound a condition slightly weaker than the condition $\alpha(H) > \rho s$. Namely, using this scheme, we are able to bound the probability for which $\alpha(H) > \delta\rho s$ for sufficiently large constants δ. This result is used to prove Theorem 3.11 of section 3. In section 5.3 we refine our scheme and obtain the main result of this section.

THEOREM 5.4. *Let G be a $\langle\rho,\varepsilon\rangle$-connected graph. Let H be a random sample of G of size s. For any constant $c_1 > 0$ there exists a constant $c_2 > 0$ (depending on c_1 alone) s.t.*

1. *if $s \geq c_2 \frac{\rho^4}{\varepsilon^3} \log\left(\frac{\rho}{\varepsilon}\right)$, then the probability that H has an independent set of size $> \rho s$ is at most $e^{-c_1 \frac{\rho}{\varepsilon}}$;*
2. *if G is pairwise $\langle\rho,\varepsilon\rangle$-connected, and $s \geq c_2 \frac{\rho^5}{\varepsilon^3} \log\left(\frac{\rho}{\varepsilon}\right) \log(\frac{1}{\rho})$, then the probability that H has an independent set of size $> \rho s$ is at most $e^{-c_1 \frac{\rho^2 \log(1/\rho)}{\varepsilon}}$.*

5.1. The naive scheme. Let $G = (V, E)$ be a $\langle\rho,\varepsilon\rangle$-connected graph (pairwise or not). In this section we study the probability that a random subset R of V of size k is an independent set. We then use this result to bound the probability that a random subset H of G of size s has a large independent set.

We would like to bound (from above) the probability that R induces an independent set. Let $\{r_1, \ldots, r_k\}$ be the vertices of R. Consider choosing the vertices of R one by one such that at each step the random subset chosen so far is $R_i = \{r_1, \ldots, r_i\}$. Assume that at some stage R_i is an independent set. We would like to show (with high probability) that after adding the remaining vertices of $\{r_{i+1}, \ldots, r_k\}$ to R_i, the final set R will not be an independent set.

Let $I(R_i)$ (for *independent*) be the set of vertices in V which are not adjacent to any vertices in R_i, and let $N(R_i)$ be the set of vertices that are adjacent to a vertex in R_i. Consider the next random vertex $r_{i+1} \in R$. If r_{i+1} is chosen from $N(R_i)$, then R_{i+1} is no longer an independent set (implying that neither is R), and we view this round as a success. Otherwise, r_{i+1} happens to be in $I(R_i)$, and R_{i+1} is still an independent set. But if r_{i+1} also happens to have many neighbors in $I(R_i)$, then adding it to R_i will substantially reduce the size of $I(R_{i+1})$, which works in our favor. This later case is also viewed as a successful round regarding R_i.

Motivated by the discussion above, we continue with the following definitions. As before, let $G = (V, E)$ be a $\langle\rho,\varepsilon\rangle$-connected graph (pairwise or not), let $R = \{r_1, \ldots, r_k\}$ be a set of vertices in V, and let $R_i = \{r_1, \ldots, r_i\}$. Each subset R_i of V defines the following partition (LI_i, HI_i, N_i) of V:

- Let I_i be the vertices that are not adjacent to any vertex in R_i (notice that it may be the case that $R_i \cap I_i \neq \phi$). I_i is now partitioned into two parts: vertices in I_i which have *low* degree, denoted as the set LI_i, and vertices of *high* degree, denoted as HI_i. Namely, LI_i is defined to be the ρn vertices of I_i with minimal degree (in the subgraph induced by I_i), and HI_i is defined to be the remaining vertices of I_i. Ties are broken arbitrarily or in *favor* of vertices in R_i (namely, vertices in R_i are placed in I_i before other vertices of identical degree). If it is the case that $|I_i| \leq \rho n$, then LI_i is defined to be I_i,

and HI_i is defined to be empty.

- N_i is defined to be the remaining vertices of V (namely, the vertices that share an edge with some vertex in R_i).

We define the partition corresponding to $R_0 = \phi$ as (LI_0, HI_0, NI_0), where LI_0 are ρn vertices of G of minimal degree, HI_0 are the remaining vertices of G, and $N_0 = \phi$.

Notice, using this notation, that the subset R_i is an independent set iff $R_i \cap N_i = \phi$, or equivalently, $R_i \subseteq I_i$. Moreover, in this case $R_i \subseteq LI_i$. (All vertices of R_i have degree 0 in the subgraph induced by I_i.) Furthermore, each vertex r_i in an independent set $R = R_k = \{r_1, \ldots, r_k\}$ satisfies $r_i \in I_{i-1}$.

We are now ready to bound the probability that a random subset $R = \{r_1, \ldots, r_k\}$ of G is independent. Let $R_i = \{r_1, \ldots, r_i\}$, and let (LI_i, HI_i, N_i) be the corresponding partitions of V defined by R_i. Consider the case in which R is an independent set. As mentioned above, this happens iff for every i the vertex r_i is chosen to be independent from the subset R_{i-1}, or in other words, $r_i \in I_{i-1} = LI_{i-1} \cup HI_{i-1}$. We would like to show that this happens with small probability (if k is large enough).

Initially, the subset I_0 is large (the entire vertex set V), and it gets smaller and smaller as we proceed in the choice of vertices in R. Each vertex in $r_i \in HI_{i-1}$ reduces the size of I_{i-1} substantially, while each vertex in LI_{i-1} may only slightly change the size of I_{i-1}. In the following, we show that there cannot be many vertices $r_i \in R$ that happen to fall into HI_{i-1} (as each such vertex reduces the size of I_{i-1} substantially). We thus turn to considering vertices r_i that fall in LI_{i-1}. (There are almost k such vertices.) The size of LI_i is bounded by ρn. Hence, the probability that $r_i \in LI_i$ is bounded by ρ (by our definitions $R_{i-1} \subseteq LI_{i-1}$ and the vertex r_i is random in $V \setminus R_{i-1}$). This implies that the probability that R is an independent set is roughly bounded by ρ^k. Details follow.

LEMMA 5.5. *Let G be a $\langle \rho, \varepsilon \rangle$-connected graph. Let $R = \{r_1, \ldots, r_k\}$ be a set in G. The number of vertices r_i that satisfy $r_i \in HI_{i-1}$ is bounded by $t = \frac{\rho}{\varepsilon}$. If G is pairwise $\langle \rho, \varepsilon \rangle$-connected, then the number of vertices r_i that satisfy $r_i \in HI_{i-1}$ is bounded by $t = \frac{2\rho^2 \lceil \log(1/\rho) \rceil}{\varepsilon}$.*

Proof. We start with the following claim.

CLAIM 5.6. *Let R_i be as defined above, and let (LI_i, HI_i, N_i) be its corresponding partition. Let $I_i = LI_i \cup HI_i$. If G is $\langle \rho, \varepsilon \rangle$-connected, then every vertex in HI_i has degree at least $\frac{\varepsilon}{\rho} n$ (in the subgraph induced by I_i). If G is pairwise $\langle \rho, \varepsilon \rangle$-connected, then every vertex in HI_i has degree at least $\frac{\varepsilon}{2\rho^2}|I_i|$ (in the subgraph induced by I_i).*

Proof. Assume that $|I_i| = \alpha \rho n$ for some $\alpha \geq 1$ (otherwise the set HI_i is empty, and the claim holds). Notice that this implies $|LI_i| = \rho n$. For the first part of our claim, recall by the definition of $\langle \rho, \varepsilon \rangle$-connected graphs that $E(LI_i, LI_i) \geq \varepsilon n^2$. Hence, we conclude that there exists a vertex in LI_i of degree at least $\frac{\varepsilon}{\rho} n$ (in the subgraph induced by LI_i). The set $LI_i \subseteq I_i$, and thus also, in the subgraph induced by I_i, there exists a vertex in LI_i of degree at least $\frac{\varepsilon}{\rho} n$. As LI_i are the vertices of minimal degree in I_i, we conclude the first part of our assertion.

For the second part, let $I_i = X_1 \cup X_2 \cup \cdots \cup X_\ell$, where $\{X_1, \ldots, X_\ell\}$ is a partition of I_i into ℓ sets in which the size of X_j for all $j \neq \ell$ is ρn. Notice that $\ell = \lfloor \alpha \rfloor + 1$. For each $v \in I_i$ let $d_v(I_i)$ be the degree of v in the subgraph induced by I_i.

In this case our graph G is pairwise $\langle \rho, \varepsilon \rangle$-connected. This implies that the value of $E(LI_i, X_j)$ for each j (except $j = \ell$) is at least εn^2. Hence, $\sum_{v \in LI_i} d_v(I_i)$ is at least $\lfloor \alpha \rfloor \varepsilon n^2 \geq \frac{\alpha}{2} \varepsilon n^2$. This implies that LI_i must include a vertex v with degree $d_v(I_i) \geq \alpha \frac{\varepsilon}{2\rho} n$. As LI_i are the vertices of minimal degree in I_i, we conclude our assertion. ∎

Now to prove our lemma, consider the subsets $R_i = \{r_1, \ldots, r_i\}$ and their corresponding partitions (LI_i, HI_i, N_i). Let $I_i = LI_i \cup HI_i$. Let $N(r_i)$ be the vertices adjacent to r_i in I_{i-1}. We would like to bound the number of vertices r_i that are in HI_{i-1}. We start with the case in which G is $\langle \rho, \varepsilon \rangle$-connected. Consider a vertex r_i in HI_i. By Claim 5.6, its degree in I_{i-1} is $|N(r_i)| \geq \frac{\varepsilon}{\rho} n$. Each vertex $r_i \in HI_{i-1}$ increases the size of N_{i-1} by at least $|N(r_i)|$. Initially, N_0 is empty, and after r_k is chosen, $|N_k| \leq n$. We conclude that there are at most $\frac{\rho}{\varepsilon}$ vertices r_i in R which are in HI_{i-1}. This bound can be further improved by a factor of approximately ρ to $\frac{2\rho^2 \lceil \log(1/\rho) \rceil}{\varepsilon}$ using tighter analysis when G is also pairwise $\langle \rho, \varepsilon \rangle$-connected; details follow.

Let $x \geq 0$ be an integer, and let $S_x = \{i \mid r_i \in HI_{i-1} \text{ and } |I_{i-1}| \in [\frac{n}{2^{x+1}}, \frac{n}{2^x})\}$. We would like to bound the size of S_x for all possible values of x. We start by considering values of x between 0 and $\lceil \log(1/\rho) \rceil - 1$. Consider a vertex r_i in which $i \in S_x$. That is, $r_i \in HI_{i-1}$ and $\frac{n}{2^x} \geq |I_{i-1}| > \frac{n}{2^{x+1}}$. By Claim 5.6, the degree of r_i in I_{i-1} is $|N(r_i)| \geq \frac{\varepsilon}{2\rho^2} |I_{i-1}| \geq \frac{\varepsilon}{2\rho^2} \frac{n}{2^{x+1}}$. Each vertex r_i in which $i \in S_x$ increases the size of N_{i-1} by at least $|N(r_i)|$. For such vertices, N_{i-1} is of size at least $n - \frac{n}{2^x}$ and at most $n - \frac{n}{2^{x+1}}$. We conclude that $|S_x|$ is of size at most $\frac{2\rho^2}{\varepsilon}$.

For $x \geq \lceil \log(1/\rho) \rceil$, the set S_x is a subset of $\{i \mid r_i \in HI_{i-1} \text{ and } |I_{i-1}| \leq \rho n\}$. Recall that $HI_{i-1} = \phi$ whenever $|I_{i-1}| \leq \rho n$. This implies that $S_x = \phi$ in these cases. In sum, we conclude that $\sum_x |S_x| \leq \frac{2\rho^2 \lceil \log(1/\rho) \rceil}{\varepsilon}$, which concludes our proof. ∎

THEOREM 5.7. *Let G be a $\langle \rho, \varepsilon \rangle$-connected graph (pairwise or not). Let t be as in Lemma 5.5. Let $k \geq 2t$. The probability that k random vertices of G induce an independent set is at most*

$$\rho^k \left(\frac{ek}{t\rho} \right)^t.$$

Proof. Let $R = \{r_1, \ldots, r_k\}$ be a set of k random vertices. As mentioned previously, the probability that $r_i \in LI_{i-1}$ is at most ρ. This follows from the fact that (1) the size of LI_{i-1} is at most ρn, (2) $R_{i-1} \subseteq LI_{i-1}$ (by our definitions), and (3) the vertex r_i is random in $V \setminus R_{i-1}$.

Now in order for R to be an independent set, every vertex r_i of R must be in the set I_{i-1}. Furthermore, by Lemma 5.5 all but t vertices r_i of R must satisfy $r_i \in LI_{i-1}$. Hence, the probability that R is an independent set is at most

$$\binom{k}{t} \rho^{k-t} \leq \left(\frac{ke}{t} \right)^t \rho^{k-t} = \rho^k \left(\frac{ek}{t\rho} \right)^t. \qquad \blacksquare$$

Let δ be a large constant. We now use Theorem 5.7 to bound the probability that a random subset H of G of size s has an independent set of size $> \delta \rho s$. The result is the following Corollary 5.8, which will be used in section 5.2 to prove Theorem 3.11. In section 5.3 we refine our proof techniques and *get rid* of the parameter δ. That is, we bound the probability that a random subset H of G of size s has an independent set of size $> \rho s$.

COROLLARY 5.8. *Let G be a $\langle \rho, \varepsilon \rangle$-connected graph (pairwise or not). Let t be as in Lemma 5.5. Let H be a random sample of G of size s. Let $\delta > e$, and let c be a sufficiently large constant. If $s \geq ct \frac{\log(1/\rho)}{\rho}$, then the probability that $\alpha(H) > \delta \rho s$ is at most $\left(\frac{e}{\delta}\right)^{\Omega(\delta \rho s)}$.*

Proof. Let $k = \delta \rho s$. Using Theorem 5.7 and the fact that a subset R of H is random in G, the probability that there is an independent set R in H of size k is at

most

$$\binom{s}{k}\rho^k\left(\frac{ek}{t\rho}\right)^t \le \left[\left(\frac{e}{\delta}\right)^{\frac{k}{t}}\frac{ek}{t\rho}\right]^t \le \left(\frac{e}{\delta}\right)^{\Omega(k)}.$$

In the last inequality we have used the fact that $\frac{k}{t}$ is greater than $c\log(1/\rho)$ for a sufficiently large constant c. □

5.2. Proof of Theorem 3.11. We would now like to show that the analysis presented in section 5.1 also holds for our continuous graph G_k of section 3. Namely, we would like to prove the following analogue of Corollary 5.8.

THEOREM 3.11 (restated). *Let a, k, $\varepsilon(a)$ be defined as in Theorem* 3.6 *or* 3.8. *Let H be a random sample of s vertices of G_k (according to the uniform distribution on S^{d-1}). Let c be a sufficiently large constant. If $s \ge c\frac{\rho(a)}{\varepsilon(a)}\log^2(1/\rho(a))$, then the probability that $\alpha(H) > e^2\rho(a)s$ is at most* 1/4.

Proof. Let $H = \{h_1, \ldots, h_s\}$ be s random points of the unit sphere (that is, H is a random subset of G_k of size s). Let $R_k = \{r_1, \ldots, r_k\}$ be a subset of H of size k. Finally, for $i \in \{1, \ldots, k\}$ let $R_i = \{r_1, \ldots, r_i\}$ and (LI_i, HI_i, N_i) be its corresponding partition (as defined in section 5.1). For each i the subsets LI_i, HI_i, and N_i are measurable (this follows from the fact that the neighborhood of each vertex is a cap of S^{d-1}). Hence, the proofs of Lemma 5.5 and Theorem 5.7 hold under Definitions 3.1, 3.2, and 3.9 of section 3. This suffices to conclude our assertion. □

5.3. An enhanced analysis. Let $G = (V, E)$ be a $\langle\rho, \varepsilon\rangle$-connected graph (pairwise or not), and let $H = \{h_1, \ldots, h_s\}$ be a set of random vertices of size s in V. In the previous section we presented a bound on the probability that $\alpha(H) > \delta\rho s$ for large constant values of δ. In this section we enhance our analysis and bound the probability that $\alpha(H) > \rho s$ (namely, we get rid of the additional parameter δ).

Recall our proof technique from section 5.1. We started by analyzing the probability that a subset R of H of size k is an independent set. Afterwards we bounded the probability that $\alpha(H) > \delta\rho n$ by using the standard union bound on all subsets R of H of size greater than $k = \delta\rho n$. In this section we enhance the first part of this scheme by analyzing the probability that a subset R of H of size k is a *maximum* independent set in H (rather than just an independent set of H). Then, as before, using the standard union bound on all large subsets R of H, we bound the probability that $\alpha(H) > \rho s$. We show that taking the maximality property of R into account will suffice to prove Theorem 5.4.

Let $H = \{h_1, \ldots, h_s\}$ be s random vertices in G. We would like to analyze the probability that a given subset R of H of size k is a *maximum* independent set. Recall (section 5.1) that the probability that R is an independent set is bounded by approximately ρ^k. An independent set R is a maximum independent set in H only if adding any other vertex in H to R will yield a set which is no longer independent. Let $R = R_k$ be an independent set, and let (LI_k, HI_k, N_k) be the partition (as defined in section 5.1) corresponding to R. Consider an additional random vertex h from H. The probability that $R \cup h$ is no longer an independent set is approximately $|N_k|/n$ (here we assume that $|R|$ is small compared to n). The probability that for every $h \in H \setminus R$ the subset $R \cup \{h\}$ is no longer independent is thus $\simeq (|N_k|/n)^{s-k}$. Hence, the probability that a given subset R of H of size k is a *maximum* independent set is bounded by approximately $\rho^k\,(|N_k|/n)^{s-k}$. This value is substantially smaller than ρ^k iff $|N_k|$ is substantially smaller than n. We conclude that it is in our favor to somehow ensure that $|N_k|$ is not too large. We do this in an artificial manner.

Let $R = \{r_1, \ldots, r_k\}$ be an independent set, let $R_i = \{r_1, \ldots, r_i\}$, and let (LI_i, HI_i, N_i) be the partition (as defined in section 5.1) corresponding to R_i. Roughly speaking, in section 5.1, every time a vertex r_i was chosen, the subset N_i was updated. If r_i was chosen in HI_{i-1}, then N_{i-1} grew substantially, and if r_i was chosen in LI_{i-1}, the subset N_{i-1} was only slightly changed. We would like to change the definition of the partition (LI_i, HI_i, N_i) corresponding to R_i to ensure that N_i is always substantially smaller than n. This cannot be done unless we relax the definition of N_i. In our new definition, N_i will no longer represent the entire set of vertices adjacent to R_i; rather, N_i will include only a subset of vertices adjacent to R_i (a subset which is substantially smaller than n). Namely, in our new definition of the partition (LI_i, HI_i, N_i) the set N_{i-1} is changed only if r_i was chosen in HI_{i-1}. In the case in which $r_i \in LI_{i-1} \cup N_{i-1}$, we do not change N_{i-1} at all. As we will see, such a definition will imply that $|N_i| \leq (1-\rho)s$, which will now suffice for our proof.

A new partition. Let $H = \{h_1, \ldots, h_s\}$ be a subset of V. Let $R_i = \{r_1, \ldots, r_i\}$ be a subset of H of size i. Each such subset R_i defines a partition (LI_i, HI_i, N_i) of V. As before, let $I_i = LI_i \cup HI_i$.

1. Initially $R_0 = \phi$, LI_0 is the ρn vertices in V of minimal degree (in V), $HI_0 = V \setminus LI_0$, and $N_0 = \phi$. In the above, ties are broken by an assumed ordering on the vertices in V.
2. Let (LI_i, HI_i, N_i) be the partition corresponding to R_i, and let r_{i+1} be a new random vertex. Let $R_{i+1} = R_i \cup \{r_{i+1}\}$; then we define the partition $(LI_{i+1}, HI_{i+1}, N_{i+1})$. Let $N(r_{i+1})$ be the neighbors of r_{i+1} in I_i. We consider the following cases:
   - If $r_{i+1} \in LI_i$, then the partition corresponding to R_{i+1} will be exactly the partition corresponding to R_i, namely, $LI_{i+1} = LI_i$, $HI_{i+1} = HI_i$, and $N_{i+1} = N_i$. Notice that this implies that N_{i+1} no longer represents all neighbors of R_{i+1}. There may be vertices adjacent to R_{i+1} which are in I_{i+1}.
   - If $r_{i+1} \in HI_i$, then we consider two subcases:
     - If $|N_i \cup N(r_{i+1})| \leq (1-\rho)n$, then LI_{i+1}, HI_{i+1}, and N_{i+1} are defined as in section 5.1. Namely, $N_{i+1} = N_i \cup N(r_{i+1})$. I_{i+1} is defined to be $V \setminus N_{i+1}$. LI_{i+1} is defined to be the ρn vertices of I_{i+1} with minimal degree (in the subgraph induced by I_{i+1}), and HI_{i+1} is defined to be the remaining vertices of I_{i+1}. Ties are broken by the assumed ordering on V.
     - If $|N_i \cup N(r_{i+1})| > (1-\rho)n$, then let $\hat{N}(r_{i+1})$ be the first (according to the assumed ordering on V) $(1-\rho)n - |N_i|$ vertices in $N(r_{i+1})$, and set $N_{i+1} = N_i \cup \hat{N}(r_{i+1})$. Furthermore, set LI_{i+1} to be the remaining ρn vertices of G, and HI_i to be empty. Notice that in this case, $|N_{i+1}|$ is of size exactly $(1-\rho)n$.
   - If $r_{i+1}, \in N_i$ then, once again, the partition corresponding to R_{i+1} will be exactly the partition corresponding to R_i.

A few remarks are in order. First, it is not hard to verify that the definition above implies the following claim.

CLAIM 5.9. *Let $i \in \{1, \ldots, k\}$. The partitions (LI_i, HI_i, N_i) corresponding to R_i as defined above satisfy* (a) $I_i \subseteq I_{i-1}$, (b) $N_{i-1} \subseteq N_i$, (c) $|N_i| \leq (1-\rho)n$, (d) $|LI_i| = \rho n$, (e) *that the set LI_i is the ρn vertices of minimal degree in I_i.*

Second, due to the iterative definition of our new partition, the partitions (LI_i, HI_i, N_i) corresponding to the subsets R_i depend strongly on the specific ordering of

the vertices in R_i. Namely, in contrast to the partitions used in section 5.1, a single subset R with two different orderings may yield two different partitions. For this reason, in the remainder of this section we will assume that the vertices of H are chosen one by one. This will imply an ordering on H and on any subset R of H. The partitions we will study will correspond to these orderings only.

Finally, in section 5.1, an (ordered) subset $R = \{r_1, \ldots, r_k\}$ was independent iff for all i, $r_i \in I_{i-1}$ (according to the definition of I_{i-1} appearing in section 5.1). In this section, if R is independent, then it still holds that for all i, $r_i \in I_{i-1}$. However, it may be the case that for all i, $r_i \in I_{i-1}$, but R is not an independent set. In the remainder of this section, we call ordered subsets R for which for all i, $r_i \in I_{i-1}$, *free* sets. We analyze the probability that a random ordered subset H of V of size s does not have any free sets of size larger then ρs. This implies that H does not include any independent sets of size ρs.

DEFINITION 5.10. *An ordered subset $R_i = \{r_1, \ldots, r_i\}$ is said to be free if it is the case that $r_j \in I_{j-1}$ for all $j \leq i$.*

CLAIM 5.11. *Let $H = \{h_1, \ldots, h_s\}$ be an ordered set of vertices in a (pairwise) $\langle \rho, \varepsilon \rangle$-connected graph G. If $\alpha(H) > \rho s$, then the maximum free set in H (w.r.t. the ordering implied by H) is of size $> \rho s$.*

Proof. Let I be an independent set of size $> \rho s$ in H. It is not hard to verify that I (under the ordering implied by H) is a free set. We conclude that the maximum free set R in H (ordered by the ordering implied by H) is of size $> \rho s$. ∎

Claim 5.11 implies that to prove Theorem 5.4 it suffices to analyze the maximum free set $R \subseteq H$. Moreover, the only ordered subsets R that we need to consider are those ordered by the ordering implied by H. We now turn to proving Theorem 5.4. Roughly speaking, we start by analyzing the probability that a random subset R is a free set. We then analyze the probability that a given subset R in H is a maximum free set. Finally, we use the union bound on all subsets R of H of size $> \rho s$ to obtain our results.

In the remainder of this section, we will assume that the subset H is chosen from G randomly *with repetitions*. That is, H is a random multiset of size s. Our results (with minor modifications) apply also to the case in which H is a random subset of G (and not a multiset) if the size of H is not very large (here we assume that $|H| \ll \sqrt{n}$). As in such cases, a set H of size s which is randomly chosen from V with repetitions will not include the same vertex twice (with high probability).

We start by stating the following lemmas, which are analogous to Lemma 5.5 and Theorem 5.7 from section 5.1. The main difference between the lemmas below (and their proofs) and those of the previous section is in the definition of the partition (LI_i, HI_i, N_i) and in the fact that they address free sets instead of independent sets. Proof of the lemmas is omitted.

LEMMA 5.12. *Let G be a $\langle \rho, \varepsilon \rangle$-connected graph. Let $R = \{r_1, \ldots, r_k\}$ be an ordered set in G of size k. The number of vertices r_i which satisfy $r_i \in HI_{i-1}$ is bounded by $t = \frac{\rho}{\varepsilon}$. If G is pairwise $\langle \rho, \varepsilon \rangle$-connected, then the number of vertices r_i which satisfy $r_i \in HI_{i-1}$ is bounded by $t = \frac{2\rho^2 \lceil \log(1/\rho) \rceil}{\varepsilon}$.*

LEMMA 5.13. *Let G be a $\langle \rho, \varepsilon \rangle$-connected graph (pairwise or not). Let t be as in Lemma 5.12. Let $k \geq 2t$. Let $R = \{r_1, \ldots, r_k\}$ be k random vertices of G. The probability that R induces a free set is at most*

$$\rho^k \left(\frac{ek}{t\rho} \right)^t.$$

We now address the probability that a random subset R of H is a maximum free set. We will then use the union bound on all subsets R of H of size $> \rho s$ to obtain our results.

LEMMA 5.14. *Let G be a $\langle \rho, \varepsilon \rangle$-connected graph (pairwise or not). Let t be as in Lemma 5.12. Let $k \geq 2t$. Let H be an ordered random sample of G of size $s \geq k$. The probability that a given subset R of H is a maximum free set is at most*

$$\rho^k \left(\frac{ek}{t\rho}\right)^t (1-\rho)^{s-k}.$$

Proof. Let $R = \{r_1, \ldots, r_k\}$ (ordered by the ordering induced by H). The set R is a maximum free set in H only if (a) R is free and (b) for each vertex $h \in H$ which is not in R, the ordered set $R^+ = \{r_1, \ldots, r_j, h, r_{j+1}, \ldots, r_k\}$ is not free. Here the index j is such that r_j appears before h in the ordering of H, and r_{j+1} appears after h (i.e., R^+ is ordered according to the ordering of H).

The probability that R is free has been analyzed in Lemma 5.13. It is left to analyze the probability that R^+ is not free for every vertex $h \notin R$, given that R is free. Consider a vertex $h \in H$ which is not in R, and let $R^+ = \{r_1, \ldots, r_j, h, r_{j+1}, \ldots, r_k\}$.

CLAIM 5.15. *Let $R = \{r_1, \ldots, r_k\}$ be a free set and $R^+ = \{r_1, \ldots, r_j, h, r_{j+1}, \ldots, r_k\}$. Let the partition corresponding to $R_j = \{r_1, \ldots, r_j\}$ be (LI_j, HI_j, N_j). If $h \in LI_j$, then R^+ is also a free set.*

Proof. We will use the following notation. Let $R_i = \{r_1, \ldots, r_i\}$ denote the first i vertices of R, and let (LI_i, HI_i, N_i) be its corresponding partition. For $i > j$, let $R_i^+ = \{r_1, \ldots, r_j, h, r_{j+1}, \ldots, r_i\}$ denote the first $i+1$ vertices of R^+, and let (LI_i^+, HI_i^+, N_i^+) be its corresponding partition. Finally, let R_h^+ denote the subset $\{r_1, \ldots, r_j, h\}$ and (LI_h^+, HI_h^+, N_h^+) be its corresponding partition.

We would like to prove that R^+ is free. That is, we would like to show (a) that $r_i \in I_{i-1}$ for each $i \leq j$, (b) that $h \in I_j$, (c) that $r_{j+1} \in I_h^+$, and (d) that $r_i \in I_{i-1}^+$ for $i \geq j+2$. Recall that R is free, and thus $r_i \in I_{i-1}$ for all $i \in \{1, \ldots, k\}$.

The first assertion follows from the fact that the first j vertices of R and R^+ are identical. The second follows from the assumption that $h \in LI_j$. For the third assumption, notice (as $h \in LI_j$) that the partition corresponding to $R_h^+ = \{r_1, \ldots, r_j, h\}$ is equal to the partition corresponding to $R_j = \{r_1, \ldots, r_j\}$. This follows from our definition of the partition (LI_h^+, HI_h^+, N_h^+). As $r_{j+1} \in I_j$, we conclude that $r_{j+1} \in I_h^+$.

For the final assertion, observe that for any $i \geq j+1$, the partition corresponding to R_i^+ is equal to the partition corresponding to R_i. This can be seen by induction (on i). We start with the partitions corresponding to R_{j+1} and R_{j+1}^+. The partition $(LI_{j+1}, HI_{j+1}, N_{j+1})$ is defined uniquely by the partition corresponding to R_j and the vertex r_{j+1}. Similarly, the partition $(LI_{j+1}^+, HI_{j+1}^+, N_{j+1}^+)$ is defined uniquely by the partition corresponding to R_h^+ and the vertex r_{j+1}. As the partition corresponding to R_h^+ is equal to the partition corresponding to R_j, we conclude that the same hold for the partitions corresponding to R_{j+1} and R_{j+1}^+. The inductive step is done similarly. The partition corresponding to R_i (R_i^+) is defined uniquely by the partition corresponding to R_{i-1} (R_{i-1}^+) and the vertex r_i. As the partition corresponding to R_{i-1} equals that corresponding to R_{i-1}^+, we conclude our claim. As R is free, $r_i \in I_{i-1}$ for every $i \geq j+2$. This implies also that $r_i \in I_{i-1}^+$, which proves the final assertion. ∎

Claim 5.15 implies that the probability that $R^+ = \{r_1, \ldots, r_j, h, r_{j+1}, \ldots, r_k\}$ is not free, given that R is free, is at most $(1-\rho)$ (recall that the set LI_j is of size exactly ρn). This holds independently for every vertex h in $H \setminus R$. We conclude that

the probability that R is a maximum free subset of H is at most the probability that R is free times $(1-\rho)^{s-k}$. ∎

We now turn to analyzing the probability that a random ordered subset H of G of size s has a free set of size larger than ρs. We follow the line of analysis given in section 5.1 and analyze the probability that H has a free set of size larger than $\delta\rho s$ for any $\delta > 1$. We then get rid of the factor δ to obtain our main theorem of this section.

COROLLARY 5.16. *Let G be a $\langle\rho,\varepsilon\rangle$-connected graph (pairwise or not). Let t be as in Lemma* 5.12. *Let H be a random sample of G of size s. Let $\delta > 1$, and let c be a sufficiently large constant. Let $\Gamma = \ln\delta - \frac{\delta-1}{\delta}$. If $s \geq \frac{ct}{\rho\Gamma}\left(\log(1/\rho) + \log(1+1/\Gamma)\right)$, then the probability that H has a free set of size $> \delta\rho s$ is at most*

$$s\left(\frac{1}{e^{\Gamma}}\right)^{\Omega(\delta\rho s)}.$$

Proof. Let $k = \delta\rho s$, let $\delta' > \delta$, and let $k' = \delta'\rho s > k$. Using Lemma 5.14, the probability that there is a maximum free set R in H of size k' is at most

$$\begin{aligned}
\sum_{k'>k}\binom{s}{k'}\rho^{k'}\left(\frac{ek'}{t\rho}\right)^{t}(1-\rho)^{s-k'} &\leq \sum_{k'>k}\left(\frac{ek'}{t\rho}\right)^{t}\frac{s^s(1-\rho)^{s-k'}}{{k'}^{k'}(s-k')^{s-k'}}\rho^{k'} \\
&\leq \sum_{k'>k}\left(\frac{ek'}{t\rho}\right)^{t}\frac{e^{k'\frac{\delta'-1}{\delta'}}}{{\delta'}^{k'}} \\
&= \sum_{k'>k}\left[\frac{ek'}{t\rho}\left(\frac{1}{e^{\ln\delta'-\frac{\delta'-1}{\delta'}}}\right)^{\frac{k'}{t}}\right]^{t} \\
&\leq \sum_{k'>k}\left[\frac{ek'}{t\rho}\left(\frac{1}{e^{\Gamma}}\right)^{\frac{k'}{t}}\right]^{t} \\
&\leq \sum_{k'>k}\left(\frac{1}{e^{\Gamma}}\right)^{\Omega(k')} \leq s\left(\frac{1}{e^{\Gamma}}\right)^{\Omega(k)}.
\end{aligned}$$

We use the facts that $\frac{k'}{t}$ is greater than both $c\frac{1}{\Gamma}\log(1+1/\Gamma)$ and $c\frac{1}{\Gamma}\log(1/\rho)$ for a sufficiently large constant c, and that Γ is an increasing function of δ (for $\delta > 1$). ∎

It remains to get rid of the additional parameter δ of Corollary 5.16 (namely, to analyze the probability that $\alpha(H) > \rho s$).

LEMMA 5.17. *If a given graph G is (pairwise) $\langle\rho,\varepsilon\rangle$-connected, then G is also (pairwise) $\langle\rho(1-\frac{\varepsilon}{4\rho^2}),\frac{\varepsilon}{2}\rangle$-connected.*

Proof. We present proof for the case in which G is $\langle\rho,\varepsilon\rangle$-connected; a similar proof holds for the case in which G is pairwise $\langle\rho,\varepsilon\rangle$-connected. Let A be some subset of G of size $\rho(1-\frac{\varepsilon}{4\rho^2})n$. Let A^c be any set in $V\setminus A$ of size $\frac{\varepsilon}{4\rho}n$. It is known that the number of edges induced by the set $A\cup A^c$ is at least $\frac{\varepsilon}{2}n^2$ (notice that $|A\cup A^c| = \rho n$ and $E(A\cup A^c, A\cup A^c) \geq \varepsilon n^2$). The number of edges (in $A\cup A^c$) adjacent to vertices in A^c is bounded by $\frac{\varepsilon}{4\rho}\rho n^2 = \frac{\varepsilon}{4}n^2$. Hence, the number of edges induced by vertices in A is at least $\frac{\varepsilon n^2}{4}$, implying that $E(A,A) \geq \frac{\varepsilon}{2}n^2$. ∎

THEOREM 5.4 (restated). *Let G be a $\langle\rho,\varepsilon\rangle$-connected graph (pairwise or not). Let t be as in Lemma* 5.12. *Let H be a random sample of G of size s. Let c be a*

sufficiently large constant. If $s \geq ct\frac{\rho^3}{\varepsilon^2}\log\left(\frac{\rho}{\varepsilon}\right)$, *then the probability that* H *has an independent set of size* $> \rho s$ *is at most* $e^{-\Omega(t)}$.

Proof. By Lemma 5.17, G is also $\langle \rho(1-\frac{\varepsilon}{4\rho^2}), \frac{\varepsilon}{2}\rangle$-connected (pairwise or not). Let $\rho' = \rho(1-\frac{\varepsilon}{4\rho^2})$ and $\varepsilon' = \frac{\varepsilon}{2}$. We would like to bound the probability that H does not have any independent sets of size greater than ρs. Let $\delta = 1 + \frac{\varepsilon}{4\rho^2}$. Notice that $\delta\rho' \leq \rho$. Hence, it suffices to bound the probability that $\alpha(H) > \delta\rho' s$. This probability, in turn, is at most the probability that H has a maximum free set of size greater than $k = \delta\rho' s$ (Claim 5.11).

Let $\Gamma = \ln(\delta) - \frac{\delta-1}{\delta}$. It is not hard to verify that $\Gamma = \theta((\delta-1)^2) = \theta(\frac{\varepsilon^2}{\rho^4})$ for our value of δ. By our assumption, s is greater than or equal to

$$\begin{aligned} ct\frac{\rho^3}{\varepsilon^2}\log\left(\frac{\rho}{\varepsilon}\right) &\geq \frac{c_1 t}{\rho(\delta-1)^2}\left(\log\left(\frac{1}{\rho'}\right) + \log\left(1+\frac{1}{(\delta-1)^2}\right)\right) \\ &\geq \frac{c_2 t}{\rho'\Gamma}\left(\log\left(\frac{1}{\rho'}\right) + \log\left(1+\frac{1}{\Gamma}\right)\right), \end{aligned}$$

where in the above, c_1 and c_2 are constants closely related to c. Now, by Corollary 5.16, for our choice of s, the probability that H has a maximum free set of size greater than $k = \delta\rho' s$ is at most $s\left(\frac{1}{e^\Gamma}\right)^{\Omega(\delta\rho' s)} \leq e^{-\Omega(t)}$. ∎

Roughly speaking, Theorem 5.4 states that, given a $\langle\rho,\varepsilon\rangle$-connected graph G, a random sample H of G of size s proportional to $\frac{\rho^4}{\varepsilon^3}$ (or larger) will not have an independent set of size ρs (with high probability). This improves upon the bound of $s \simeq \frac{\rho}{\varepsilon^4}$ presented in [GGR98] both in the dependence on ρ (as $\rho < 1$) and in the dependence on ε. Moreover, we present a further improvement to $s \simeq \frac{\rho^5}{\varepsilon^3}$ if our graphs are considered to be pairwise $\langle\rho,\varepsilon\rangle$-connected. In section 6 we continue to study the minimal value of s for which $\alpha(H) < \rho s$ with high probability, and present a lower bound on the size of s which is proportional to $\frac{\rho^3}{\varepsilon^2}$.

6. Lower bounds for the testing of $\alpha(G)$. In this section we present graphs G which are $\langle\rho,\varepsilon\rangle$-connected, but with some constant probability a random sample R of G of size $s \sim \rho^3/\varepsilon^2$ is likely to have an independent set of size greater than ρs.

LEMMA 6.1. *Let* ρ *be a small constant and* $\varepsilon < \rho^2$ *s.t.* $\rho^3/\varepsilon^2 \ll n$. *For* n *large enough, there exists a graph* G *on* n *vertices for which* (a) G *is* $\langle\rho,\varepsilon\rangle$*connected, and* (b) *with constant probability (independent of* ρ *and* ε*) a random set* R *of size* $s = \frac{\rho^3}{\varepsilon^2}$ *will have an independent set of size* ρs.

Proof. Consider the graph $G = (V, E)$ in which $|V| = n$, and V consists of two disjoint sets A and $V \setminus A$, where A is an independent set of size $(1-\frac{\varepsilon}{\rho^2})\rho n$, $V \setminus A$ induces a clique, and every vertex in A is adjacent to every vertex in $V \setminus A$. On one hand, every subset of size ρn in G induces a subgraph with at least $\varepsilon n^2/2$ edges (implying that G is $\langle\rho,\varepsilon\rangle$-connected). On the other hand, let R be a random subset of V obtained by picking each vertex independently with probability $\frac{\rho^3}{\varepsilon^2 n}$. The expected size of R is $s = \frac{\rho^3}{\varepsilon^2}$. In the following we assume that R is exactly of size s; minor modifications in the proof are needed if this assumption is not made. The set $R \cap A$ is an independent set in the subgraph induced by R. The expected size of $R \cap A$ is $(1-\frac{\varepsilon}{\rho^2})\rho s$. Let $N(0,1)$ denote a standard normal variable. It can be seen using the central limit theorem (for example, [Fel66]) that, for our choice of parameters, the probability that $|R \cap A|$ deviates from its expectation by more than a square root of

its expectation is at least

$$\Pr\left[|R \cap A| > \left(1 - \frac{\varepsilon}{\rho^2}\right)\rho s + \sqrt{\rho s}\right] > \Pr\left[N(0,1) > 1\right],$$

which is some constant probability independent of ε and ρ. In such a case the size of $R \cap A$ will be greater than $(1 - \frac{\varepsilon}{\rho^2})\rho s + \sqrt{\rho s} = \rho s$ for our value of s, hence implying assertion (b) of the lemma. □

7. An alternative proof of Theorem 1.2(1). In Theorem 1.2(1), we are interested in presenting a vector k-colorable graph H in which a special relationship is satisfied between its maximum degree and its maximum independent set. It is not hard to verify that the graphs G_k presented in section 3 are far from satisfying this relationship, as the maximum degree Δ of these graphs is too large. We overcome this problem in sections 4 and 5 by considering a random (vertex-induced) subgraph of G_k. We have shown that such a subgraph will suffice for proving the three parts of Theorem 1.2.

Another method for coping with the large maximum degree Δ of G_k was suggested by Luca Trevisan (private communication). Instead of sampling vertices from G_k at random in order to obtain a sparse graph, consider sampling *edges* at random. In the following section we combine the idea of edge sampling with our results from section 3 and prove the first part of Theorem 1.2. The use of edge sampling simplifies the proof of Theorem 1.2(1) (as the analysis presented in section 5 is no longer needed). It appears that the remainder of Theorem 1.2 cannot be proven using edge sampling (as we are interested in graphs in which the maximum independent set is small with respect to the number of vertices in the graph).

To construct the graph H we will follow a three-phase plan. Our starting point will be the continuous graph G_k (of section 3), which is vector k-colorable and proven to be pairwise $\langle \rho, \varepsilon \rangle$-connected. We then define and analyze a (finite) discrete version G_k^d of G_k, which will be shown to inherit many of the properties of G_k. Namely, this discrete graph will be *almost* vector k-colorable, and will be pairwise $\langle \rho, \varepsilon \rangle$-connected. Finally, we will define H to be the graph obtained by randomly removing edges from the discrete graph G_k^d.

The discrete graph G_k^d. We now define a discrete analogue G_k^d of the continuous graph G_k from section 3. Recall that the vertex set of G_k is the d-dimensional unit sphere S^{d-1}. It is shown in [FS02] that S^{d-1} can be partitioned into $n = 2^{\theta(d^2)}$ cells of equal size and of diameter at most 2^{-d} each. Let $\mathcal{P} = \{C_1, \ldots, C_n\}$ denote the cells obtained in the above partition. The graph G_k^d will be of size n, in which each vertex $v_i \in V$ corresponds to a cell $C_i \in \mathcal{P}$. The edge set of G_k^d consists of an edge (u, v) iff there is a positive measure of edges in G_k between their corresponding cells C_u, C_v.

LEMMA 7.1. *Let A and B be subsets of G_k^d, and let A_c and B_c be the corresponding subsets of G_k.* (a) *The size of A (B) is ρn iff A_c (B_c) has measure ρ.* (b) $E(A, B) \geq E(A_c, B_c)n^2$. *The definition of $E(A, B)$ is given in Definition* 5.1 *of section* 5.

Proof. For the first part of the lemma, assume that the set A has ρn vertices; thus the corresponding subset A_c consists of ρn cells each of measure n^{-1}. We conclude that the subset A_c has measure exactly ρ. On the other hand, if A_c has measure ρ and consists of the union of k cells of measure n^{-1}, then k must be ρn. For the second part, assume that $E(A, B) = \varepsilon n^2$; then, by the fact that each edge between A and B corresponds to at most the measure of n^{-2} edges between A_c and B_c, we conclude that $E(A_c, B_c)$ is at most ε. □

THEOREM 7.2. *Let a, k, $\varepsilon(a)$ be as defined in Theorem* 3.6 *or* 3.8. *The graph G_k^d is pairwise $\langle \rho(a), \varepsilon(a) \rangle$-connected.*

Proof. Let A and B be subsets (in G_k^d) of size $\rho(a)n$. The corresponding subsets A_c and B_c of G_k are also of measure $\rho(a)$ (Lemma 7.1). By Corollary 3.10, $E(A_c, B_c) \geq \varepsilon(a)$. We conclude that $E(A, B) \geq E(A_c, B_c)n^2 \geq \varepsilon(a)n^2$. □

LEMMA 7.3. *The graph G_k^d is vector $k\big(1+\frac{ck^2}{2^d}\big)$-colorable for some constant $c > 0$.*

Proof. Recall that each cell in G_k^d has diameter at most 2^{-d}. Hence, two vertices in G_k^d are connected only if their inner product is less than $-1/(k-1) + \theta(1)2^{-d} \leq -\frac{1}{k(1+\frac{\theta(k)}{2^d})-1}$. □

By definition, the continuous graph G_k is vector k-colorable. In Lemma 7.3 we showed that the finite approximation G_k^d to G_k is *almost* vector k-colorable. In general, this does not suffice for the proof of Theorem 1.2, as we are interested in graphs which are vector k-colorable (rather than "almost vector k-colorable"). This can be fixed by starting with a continuous graph with vector coloring number slightly less than k (e.g., $k/\big(1+\frac{ck^2}{2^d}\big)$). In order to simplify our presentation, we ignore this point and consider the graph G_k^d to be exactly vector k-colorable. This is possible due to the fact that the properties of G_k are *continuous* in k. Namely, choosing d large enough, it can be seen that the multiplicative error of $(1+\frac{ck^2}{2^d})$ in the value of k does not affect the analysis appearing throughout this section.

LEMMA 7.4. *Let $\rho(\frac{1}{k-1})$ be the measure of a $\frac{1}{(k-1)}$-cap. Every vertex v in the graph G_k^d has degree $d_v \in [\frac{1}{poly(d)}\rho(\frac{1}{k-1})n, poly(d)\rho(\frac{1}{k-1})n]$.*

Proof. Consider a vertex v in G_k^d and its corresponding cell C_v. The degree of v is the number of cells in G_k that share a positive measure of edges with the cell C_v. The total measure of these cells is at least the measure of a $(\frac{1}{k-1} + \theta(2^{-d}))$-cap and at most the measure of a $(\frac{1}{k-1} - \theta(2^{-d}))$-cap. Hence, by Lemma 7.1, we conclude our theorem. □

We now prove the first part of Theorem 1.2 by considering the graph H obtained by randomly sampling the edges of G_k^d.

THEOREM 7.5. *For every constant $\gamma > 0$ and constant $k > 2$, there are infinitely many graphs H that are vector k-colorable and satisfy $\alpha(H) \leq n/\Delta_H^{1-\frac{2}{k}-\gamma}$, where n is the number of vertices in H and Δ_H is the maximum degree in H.*

Proof. Let $k > 2$ be constant. Let $\gamma > 0$ be an arbitrarily small constant. Let $a = \gamma/c$ for a sufficiently large constant c. Let $G = G_k^d = (V, E)$ be the discrete graph defined above. Let n be the size of the vertex set V of G, and let Δ be the maximum degree of G. Recall that $n = 2^{\theta(d^2)}$, where d is the dimension in which the corresponding graph G_k was defined. We will assume that the dimension d is a very large constant determined after fixing a. Finally, let $\rho = \rho(a)$.

By Lemma 7.4, all vertices in G are of degree in the range $[\frac{1}{poly(d)}\Delta, \Delta]$, where $\Delta \sim poly(d)\rho(\frac{1}{k-1})n$. By Theorem 7.2 and the proof of Claim 4.1, every subset of vertices U in G of size ρn has at least $\Delta\rho^{\frac{2(k-1)}{k-2}+\gamma}n$ edges.

Let $p = 1/(\Delta\rho^{\frac{k}{k-2}+2\gamma})$. Let H be the subgraph of G obtained by deleting each edge of G independently with probability $(1-p)$.

LEMMA 7.6. *With probability $\geq 3/4$, all vertices v of H will have degree $d_v(H)$ in the range*

$$\left[\frac{1}{poly(d)}\rho^{-\frac{k}{k-2}-2\gamma}, 2\rho^{-\frac{k}{k-2}-2\gamma}\right].$$

Proof. The expected degree $d_v(H)$ of each vertex v in H satisfies

$$d_v(H) \in \left[\frac{1}{poly(d)}\rho^{-\frac{k}{k-2}-2\gamma}, \rho^{-\frac{k}{k-2}-2\gamma}\right].$$

It is not hard to verify (using standard bounds) that with probability $\geq 3/4$ it is the case that all vertices v have degree $d_v(H)$, which does not deviate from their expectation by more than a constant fraction of their expectation. □

LEMMA 7.7. *With probability* $\geq 3/4$ *the size of the maximum independent set in* H *(*$\alpha(H)$*) is at most* ρn.

Proof. It suffices to show that every subset U of H of size ρn has at least a single edge. Using Claim 9.2, it follows that for each subset U of H, the probability that all its edges were removed is at most

$$(1-p)^{\Delta\rho^{\frac{2(k-1)}{k-2}+\gamma}n} \leq e^{-\rho^{1-\gamma}n}.$$

The number of subsets U of size ρn is

$$\binom{n}{\rho n} \leq e^{\rho n \ln n} \leq \frac{1}{4}e^{\rho^{1-\gamma}n}.$$

Applying the union bound on all subsets U of H of size ρn, we conclude our assertion. □

Now with probability at least 1/2 both Lemma 7.6 and Lemma 7.7 hold, implying our theorem. □

8. Discussion. In our work we have presented *tight* bounds on the chromatic number of vector k-colorable graphs, tight in the sense that they match the upper bounds presented in [KMS98]. Many questions still remain open.

Stronger coloring relaxations. As mentioned in the introduction and section 2, there are stronger relaxations for the minimum coloring problem that also have a geometrical interpretation. For example, one such relaxation is the well known (and extensively studied) Lovász theta function [Lov79]. It is not hard to verify that these relaxations can be used *as is* in the coloring algorithm presented in [KMS98]. One may speculate that using such stronger relaxations will yield improved coloring results. At the moment this is not known to be true.

One may consider proving that even the use of such stronger relaxations in the [KMS98] algorithm cannot yield stronger coloring results. Or in other words, one may try to extend our negative results to stronger relaxations as well. A few remarks are in place. It seems as though the techniques we use in this work do not extend to the stronger coloring relaxations presented in section 2. Our graphs (continuous and random), or to be precise, their embeddings, are not valid with respect to these stronger coloring relaxations. Therefore, in order to extend our negative results, one must change these embeddings appropriately without increasing their vector chromatic number. It seems as if changing the embedding of our graphs to satisfy these stronger coloring relaxations has a large effect on their vector chromatic number. This implies that such an approach will yield weaker negative results.

Alternatively, one may consider using our proof techniques on graphs other than the ones presented. One natural candidate is the graph $G = (V, E)$ in which the vertex set V consists of the set $\{0,1\}^n$ and two vertices are connected by an edge if their Hamming distance is equal to some prespecified value. The graph G and certain

subgraphs of G have been used in the past in the context under discussion (see, e.g., [KMS98, Fei97, GK98, Cha02]). Using our proof technique on such graphs involves the analysis of certain edge isoperimetric inequalities (analogous to those presented in Theorem 3.5). Unfortunately, little is known regarding the edge isoperimetric inequalities of the above graphs G. Such inequalities have been studied in the past [Bez02, KKL88], yielding partial results. However, these results do not suffice to extend our proof techniques. A better understanding of edge isoperimetric inequalities of these graphs is of great interest, regardless of their application to the vector coloring issue.

Property testing. In sections 5 and 6 we study the property testing paradigm with respect to the independent set problem. We present improved results on the sample size needed when testing graphs which are *far* from having large independent sets. Namely, in Theorem 5.4 we prove that if a graph G of size n is $\langle \rho, \varepsilon \rangle$-connected, then with high probability a random induced subgraph of G of size $s \sim 1/\varepsilon^3$ will not have an independent set of size ρs. (This improves upon the sample size of $s \sim 1/\varepsilon^4$ presented in [GGR98].) Moreover, in Lemma 6.1 we show that the sample size s must be of size at least $\sim 1/\varepsilon^2$ if we wish the probability of failure to be nonconstant. It would be interesting if the factor ε gap between the upper and lower bounds presented above could be settled.

9. Proof of claims.

CLAIM 9.1. *For $1 \geq p \geq 0$ and $y \geq 1$, we have that $(1-p)^y \geq 1 - py \geq (1-p)^{y+\frac{py^2}{1-py}}$.*

Proof. It is well known that $e^y \geq 1 + y$ for any real y. Hence,

$$1 - py = \frac{1}{1 + py + \frac{p^2y^2}{1-py}} \geq e^{-p(y+\frac{py^2}{1-py})} \geq (1-p)^{y+\frac{py^2}{1-py}}.$$

As for the other direction, for every $y \geq 1$

$$\frac{d}{dp}((1-p)^y - 1 + py) = y(1 - (1-p)^{y-1}) \geq 0. \qquad \square$$

CLAIM 9.2. *For all $x > 1$*

$$\left(1 - \frac{1}{x}\right)\frac{1}{e} \leq \left(1 - \frac{1}{x}\right)^x \leq \frac{1}{e}.$$

Proof. It is known that for all $x > 1$

$$\left(1 - \frac{1}{x}\right)^x \leq \frac{1}{e} \leq \left(1 - \frac{1}{x}\right)^{x-1}. \qquad \square$$

CLAIM 9.3. *Let $a > \delta > 0$ such that $a\delta \geq 2\log(d)/d$; then*

$$\rho(a) - \rho(a+\delta) \geq \left(1 - \frac{1}{d}\right)\rho(a).$$

Proof. Recall that $\frac{c}{\sqrt{d}}\left(1-a^2\right)^{\frac{d-1}{2}} \leq \rho(a) \leq \left(1-a^2\right)^{\frac{d-1}{2}}$, for some constant $c > 0$. Thus,

$$\begin{aligned}\rho(a+\delta) &\leq \left(1 - a^2 - \delta^2 - 2a\delta\right)^{\frac{d-1}{2}} \leq \left((1-a^2)(1-2a\delta)\right)^{\frac{d-1}{2}} \\ &\leq \left(1-a^2\right)^{\frac{d-1}{2}} \frac{2}{d^2} \leq \frac{\rho(a)}{d}. \qquad \square\end{aligned}$$

THEOREM 3.5 (restated). *Let $1 \geq a > 0$, and let A and B be two (not necessarily disjoint) measurable sets in V of measure $\rho(a)$. Let r be an arbitrary vertex of S^{d-1}. The minimum of $E(A,B)$ is obtained when $A = B = \mathcal{C}_a$, where $\mathcal{C}_a$ is an a-cap of measure $\rho(a)$ centered at r.*

Proof. Our proof is similar to that presented in [FS02]. We start by reviewing the *two-point symmetrization* procedure introduced in [BT76]. Let $r \in S^{d-1}$ and H be the hyperplane passing through the origin with normal r. The hyperplane H partitions S^{d-1} into three sets: points *above*, *below*, and *on* the hyperplane. Define S_0 to be the set $\{u \in S^{d-1} \mid \langle u, r\rangle = 0\}$, S^+ to be $\{u \in S^{d-1} \mid \langle u, r\rangle > 0\}$, and S^- to be $\{u \in S^{d-1} \mid \langle u, r\rangle < 0\}$. For any $x \in S^{d-1}$ denote its reflection with respect to H as $\sigma(x)$. Finally, given any (measurable) subset X of S^{d-1}, we define the *symmetrization* X^* of X w.r.t. H as follows. If x is a point in $X \cap S^-$ such that $\sigma(x) \notin X$, replace x with $\sigma(x)$. All other points $x \in X$ remain in X^*, and no other new points are added to X^*. Formally

$$X^* = X \setminus \{x \mid x \in X \cap S^-,\ \sigma(x) \notin X\} \cup \{\sigma(x) \mid x \in X \cap S^-,\ \sigma(x) \notin X\}.$$

Notice that the measure of X and X^* are identical.

Now, let Λ be the set consisting of all pairs of closed subsets in S^{d-1}. Given two closed subsets A and B of measure $\rho(a)$, let $\lambda(A,B) = \lambda \subset \Lambda$ be the set of pairs $(\alpha, \beta) \in \Lambda$ that satisfy the following:

1. $\mu(A) = \mu(\alpha)$, $\mu(B) = \mu(\beta)$;
2. for all $\varepsilon > 0$, $\mu(A_\varepsilon) \geq \mu(\alpha_\varepsilon)$ $\mu(B_\varepsilon) \geq \mu(\beta_\varepsilon)$;
3. $E(A,B) \geq E(\alpha, \beta)$,

where for any set A, the set A_ε is defined as $\{x \in S^{d-1} \mid \exists y \in A \text{ s.t. } \|x - y\| \leq \varepsilon\}$.

We start by showing that λ is closed under the two-point symmetrization technique. Next we prove that λ is a closed subset of Λ (under the Hausdorff topology). Afterwards, we show that there exists a cap C of measure $\rho(a)$ such that $(C,C) \in \lambda = \lambda(A,B)$, implying that $E(A,B) \geq E(C,C)$. This concludes our proof for subsets A and B which are closed. Finally, we turn to the general case.

LEMMA 9.4. *The set λ is closed under the two-point symmetrization procedure (w.r.t. any $r \in S^{d-1}$ and the hyperplane H passing through the origin defined by r).*

Proof. Let H be some hyperplane passing through the origin. Let (α, β) be a pair in λ. Let α^* and β^* be the sets obtained after the symmetrization procedure w.r.t. H applied to the sets α and β. It suffices to show that $(\alpha^*, \beta^*) \in \lambda$. In [Ben84] it is shown that both α^* and β^* preserve the first two properties of λ. Hence, we turn to the third property. We need to show that the subsets α^* and β^* obtained after the symmetrization procedure share fewer edges than the original sets α and β (which in turn share fewer edges than A and B). Let x^+ and y^+ be two points in S^+, and let x^- and y^- be their corresponding mirror images. Let α' be the restriction of α to these four points (that is, $\alpha' = \alpha \cap \{x^+, x^-, y^+, y^-\}$). Define β', $\alpha^{*'}$, and $\beta^{*'}$ similarly. We will show that $E(\alpha', \beta') \geq E(\alpha^{*'}, \beta^{*'})$. It is not hard to verify that this implies our assertion.

Let E' be the edge set induced by the vertices $\{x^+, x^-, y^+, y^-\}$. Recall, by the definition of our continuous graph G_k, that the edge set E consists of pairs of points x and y in S^{d-1} which satisfy $\langle x, y\rangle \leq -\frac{1}{k-1}$. This implies that E' is symmetric and has the following properties:

1. $(x^+, y^+) \in E' \leftrightarrow (x^-, y^-) \in E'$.
2. $(x^+, y^-) \in E' \leftrightarrow (x^-, y^+) \in E'$.
3. $(x^+, y^+) \in E' \rightarrow (x^+, y^-) \in E'$.

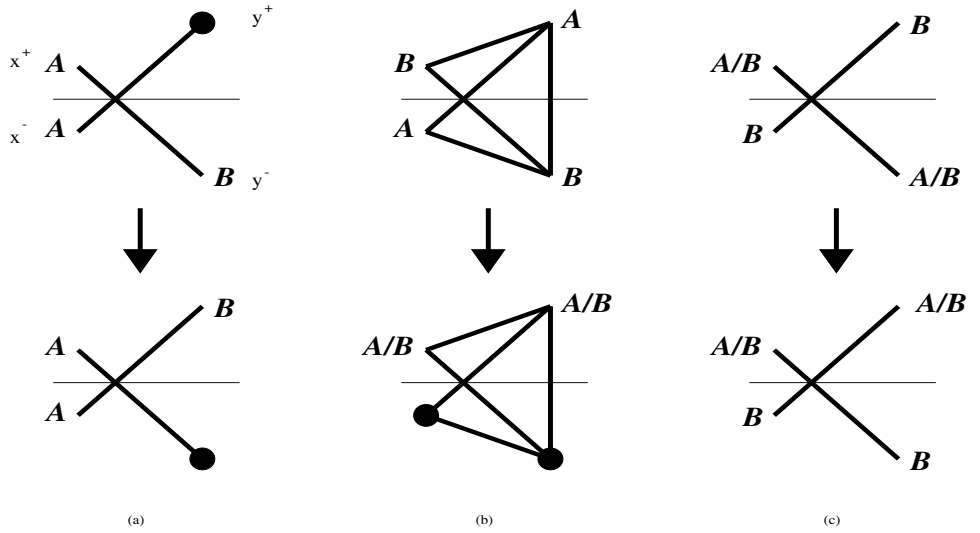


FIG. 2. *Three cases of E', α', and β' are presented. The set $\alpha' = \alpha \cap \{x^+, x^-, y^+, y^-\}$ and $\alpha^{*'}$ are presented by the letter A. The set $\beta' = \beta \cap \{x^+, x^-, y^+, y^-\}$ and $\beta^{*'}$ are presented by the letter B. A point in $\alpha' \cap \beta'$ ($\alpha^{*'} \cap \beta^{*'}$) is presented by A/B. A point in neither α nor β is represented by a solid dot. The edge set E' is depicted by solid lines. Finally, the hyperplane H is represented as a horizontal line. Each configuration is presented before (above) and after (below) the symmetrization procedure. In case* (a) *the set α' is closed under the symmetrization procedure w.r.t H and $-H$. In such cases it holds that $E(\alpha', \beta') = E(\alpha^{*'}, \beta^{*'})$. In cases* (b) *and* (c) *notice that there is a difference between the value of $E(\alpha^{*'}, \beta^{*'})$ and the number of undirected edges between the sets $\alpha^{*'}$ and $\beta^{*'}$ (for example, in case* (b)*, $E(\alpha^{*'}, \beta^{*'}) = \{(x^+, y^+), (y^+, x^+)\}$ is of size* 2*).*

Consider the case in which E' consists of the edges (x^+, y^-) and (x^-, y^+) only, the subset α' is equal to $\{x^+, x^-\}$, and the subset β' is equal to $\{y^-\}$. This case in depicted in Figure 2(a). In this specific case we have that $\alpha^{*'} = \alpha'$ and $\beta^{*'} = \{y^+\}$, implying that $E(\alpha', \beta') = E(\alpha^{*'}, \beta^{*'}) = 1$. Notice that, as α' and β' are finite sets, we measure the amount of edges between α' and β' using the discrete analogue of $E(\alpha', \beta')$ defined in Definition 5.1 of section 5 (the same goes for $\alpha^{*'}$ and $\beta^{*'}$).

There are, of course, several other cases to consider (12 edge configurations and 256 cases of different subsets α' and β'). Some of these cases are depicted in Figure 2. This large case analysis may be significantly reduced using various observations (involving the equivalence of many different cases). We have checked our assertion on the full case analysis (using a computer program). A similar proof holds when x^+ or y^+ are in S_0 (details omitted). □

LEMMA 9.5. *λ is closed in Λ.*

Proof. Let (α_n, β_n) be a sequence in λ tending to (α, β) (in the Hausdorff topology). We will show that $(\alpha, \beta) \in \lambda$. For every $\varepsilon, \delta > 0$ we have for large enough values of n that

$$\alpha_\varepsilon \subseteq (\alpha_n)_{\varepsilon+\delta}, \quad \beta_\varepsilon \subseteq (\beta_n)_{\varepsilon+\delta}; \quad \mu((\alpha_n)_{\varepsilon+\delta}) \leq \mu(A_{\varepsilon+\delta}), \quad \mu((\beta_n)_{\varepsilon+\delta}) \leq \mu(B_{\varepsilon+\delta}).$$

Sending δ to zero and using the fact that for all closed sets α, $\mu(\alpha_\varepsilon) \to \mu(\alpha)$, we conclude the second property of λ, namely that $\mu(\alpha_\varepsilon) \leq \mu(A_\varepsilon)$ and $\mu(\beta_\varepsilon) \leq \mu(B_\varepsilon)$.

For the first property, sending ε to zero, we have on one hand that $\mu(\alpha) \leq \mu(A)$ and $\mu(\beta) \leq \mu(B)$. For the other direction observe that $\mu(\alpha_\varepsilon) \geq \mu(\alpha_n) = \mu(A)$ and $\mu(\beta_\varepsilon) \geq \mu(\beta_n) = \mu(B)$ for any $\varepsilon > 0$, provided that n is large enough.

For the final property of λ, let $\theta(\varepsilon) = \mu^2(\{(x,y) \mid x \in \alpha_\varepsilon,\ y \in \beta_\varepsilon,\ (x,y) \in E\})$. Notice that $E(\alpha,\beta) \le \theta(\varepsilon)$. Let n be large enough to ensure that $\alpha_\varepsilon \subseteq (\alpha_n)_{2\varepsilon}$, $\beta_\varepsilon \subseteq (\beta_n)_{2\varepsilon}$. For such n it is the case that $\theta(\varepsilon) \le E((\alpha_n)_{2\varepsilon},(\beta_n)_{2\varepsilon})$. Furthermore, as $(\alpha_n,\beta_n) \in \lambda$ for all values of n, we have that $\mu((\alpha_n)_{2\varepsilon} \setminus \alpha_n) \le \mu(A_{2\varepsilon} \setminus A)$ and $\mu((\beta_n)_{2\varepsilon} \setminus \beta_n) \le \mu(B_{2\varepsilon} \setminus B)$. Observe that both $\mu(A_{2\varepsilon} \setminus A)$ and $\mu(B_{2\varepsilon} \setminus B)$ tend to zero as ε tends to zero. We conclude that $E((\alpha_n)_{2\varepsilon},(\beta_n)_{2\varepsilon})$ tends to $E(\alpha_n,\beta_n)$ as ε tends to zero *uniformly* in n.

Hence, given $\delta > 0$, one can find $\varepsilon > 0$ such that $E((\alpha_n)_{2\varepsilon},(\beta_n)_{2\varepsilon}) \le E(\alpha_n,\beta_n)+\delta$ for all values of n. We conclude that for n large enough

$$E(\alpha,\beta) \le \theta(\varepsilon) \le E((\alpha_n)_{2\varepsilon},(\beta_n)_{2\varepsilon}) \le E(\alpha_n,\beta_n) + \delta \le E(A,B) + \delta.$$

Since this holds for all δ, we conclude our assertion. □

LEMMA 9.6. *There exists a cap C of measure $\rho(a)$ such that $(C,C) \in \lambda$.*

Proof. Fix a point $x_0 \in S^{d-1}$, and let C be the cap centered at x_0 of measure $\rho(a)$. The function $f_1(\alpha,\beta) = \mu(C \cap \alpha)$ is upper semicontinuous on Λ.[3] Λ with the Hausdorff topology is a compact set, since λ is closed (Lemma 9.5); the function f_1 achieves its maximum over λ on some pair $(\alpha_{max},\beta_{max}) \in \lambda$. In [FS02] it is shown (using Lemma 9.4) that $\alpha_{max} = C$. This is done by showing that $f_1(\alpha^*_{max},\beta) \ge f_1(\alpha_{max},\beta)$ with equality iff $\alpha_{max} = C$, where α^*_{max} is the subset obtained by the symmetrization procedure with respect to a hyperplane H s.t. $x_0 \notin H$, and β is arbitrary.

Now consider the function $f_2(C,\beta) = \mu(C \cap \beta)$ defined on $\Lambda' = \{(C,\beta) \mid (C,\beta) \in \Lambda\}$. The function f_2 is also upper semicontinuous (this time on Λ'). Furthermore, it is shown in [FS02] that Λ' with the Hausdorff topology is a compact set and that $\lambda' = \{(C,\beta) \mid (C,\beta) \in \lambda\}$ is closed in Λ'. Therefore, the function f_2 achieves its maximum over λ' on some pair (C,β_{max}). As before, it can be seen that $\beta_{max} = C$. We conclude that $(C,C) \in \lambda$. □

As mentioned previously, Lemma 9.6 implies our theorem for subsets A and B, which are closed. The proof for general subsets A and B follows by considering any series $\{A_i\}$ and $\{B_i\}$ of closed sets that satisfy: (a) for all n, $A_n \subseteq A$ and $B_n \subseteq B$, (b) $\mu(A_n) \to \mu(A)$ and $\mu(B_n) \to \mu(B)$, and (c) for all n, $\mu(A_n) = \mu(B_n)$. Now let C, $\{C_i\}$ be caps, all centered at the same point $x \in S^{d-1}$ such that $\mu(C) = \mu(A) = \mu(B)$ and for all n, $\mu(C_n) = \mu(A_n) = \mu(B_n)$. It now holds that

$$E(A,B) \ge E(A_n,B_n) \ge E(C_n,C_n) \to E(C,C). \qquad \square$$

CLAIM 9.7. *Let a,z,δ be as in Claim 3.7. Let $b^2 = z^2 - a^2$. Let $N = \{(u_1,\ldots,u_d) \in S^{d-1} \mid u_1 = a,\ u_2 = b\}$. Let N_δ be a δ neighborhood of N (i.e., all points in S^{d-1} which are of distance less that δ from the set N). The measure of the set N_δ is at least $(1-z^2)^{\frac{d-1}{2}}$.*

Proof. Recall that $z = \sqrt{a^2+b^2}$. Let $\hat{N} = \{(u_1,\ldots,u_d) \in S^{d-1} \mid u_1 = z,\ u_2 = 0\}$ be the set obtained from N by rotating the unit sphere by an angle of $\arccos\frac{a}{z}$. Using spherical symmetry, we have that the measure of the set N_δ is equal to the measure of the δ neighborhood of the set $\hat{N}$ (which will be denoted as $\hat{N}_\delta$). Hence, in the following we will present a lower bound on the measure of $\hat{N}_\delta$. We start by computing the measure of the strip

$$S = \left\{(u_1,\ldots,u_d) \in S^{d-1} \mid u_1 \in \left[z - \frac{\delta^2}{1000}, z + \frac{\delta^2}{1000}\right]\right\}.$$

[3] $f(\alpha)$ is upper semicontinuous on Λ if for every sequence $\alpha_n \to \alpha$ in Λ it is the case that $f(\alpha)$ is at least $\limsup f(\alpha_n)$.

Using Claims 9.2 and 9.3, we conclude (for sufficiently large d) that

$$\mu(S) = \rho\left(z - \frac{\delta^2}{1000}\right) - \rho\left(z + \frac{\delta^2}{1000}\right) \geq \frac{1}{2}\rho\left(z - \frac{\delta^2}{1000}\right) \geq \frac{1}{\delta^3}\left(1 - z^2\right)^{\frac{d-1}{2}}.$$

Let γ be a constant such that $\sqrt{1-z^2} \geq |\gamma| > \delta/10$. Consider the set $R = \{(u_1, \ldots, u_d) \in S^{d-1} \mid u_1 = z,\ u_2 = \gamma\}$. Denote the $\delta^2/200$ neighborhood of R as R_γ. Using spherical symmetry, one can upper bound the measure of the set R_γ by the measure of the cap $\{(u_1, \ldots, u_d) \in S^{d-1} \mid u_1 > \sqrt{z^2 + \gamma^2} - \delta^2/200\}$, which is at most

$$\left(1 - \left(\sqrt{z^2 + \gamma^2} - \frac{\delta^2}{200}\right)^2\right)^{\frac{d-1}{2}} \leq \left(1 - z^2\right)^{\frac{d-1}{2}} \leq \delta^3 \mu(s).$$

Let $\mathcal{R}$ be the union of the sets R_γ for $|\gamma| = \delta/10, \delta/10 + \delta^2/1000, \delta/10 + 2\delta^2/1000, \ldots, \sqrt{1-z^2}$. We conclude that the measure of $\mathcal{R}$ is at most $\frac{1}{2}\mu(S)$.

Finally, each $v = (v_1, \ldots, v_d) \in S \setminus \mathcal{R}$ satisfies $v_1 \in [z - \delta^2/1000, z + \delta^2/1000]$ and $v_2 \in [-\delta/10, \delta/10]$ and thus is in $\hat{N}_\delta$. Hence $S \setminus \mathcal{R} \subseteq \hat{N}_\delta$, implying that the measure of $\hat{N}_\delta$ is at least $\frac{1}{2}\mu(S) \geq (1 - z^2)^{\frac{d-1}{2}}$. ∎

Claim 3.7 (restated). *Let a, k be as in Theorem 3.6. Let $v = (a, \sqrt{1-a^2}, 0, \ldots, 0)$ be a vertex on the boundary of $\mathcal{C}_a$. Let $N(v)$ be the set of neighbors of v. Let $z = \sqrt{a^2 + \frac{(1/(k-1)+a^2)^2}{1-a^2}}$. Finally let $\delta = c\sqrt{\frac{\log(d)}{d}}$ for a sufficiently large constant c. The measure of vertices in $N(v) \cap \mathcal{C}_a$ satisfies*

$$(1-\delta)^{\frac{d-1}{2}}(1-z^2)^{\frac{d-1}{2}} \leq \mu(N(v) \cap \mathcal{C}_a) \leq (1-z^2)^{\frac{d-1}{2}}.$$

Proof. The set $N(v) \cap \mathcal{C}_a$ is equal to

$$\left\{(u_1, \ldots, u_d) \in S^{d-1} \mid u_1 \geq a \text{ and } u_2 \leq -\frac{1/(k-1) + au_1}{\sqrt{1-a^2}}\right\}.$$

Denote $\frac{1/(k-1)+a^2}{\sqrt{1-a^2}}$ by $f(a)$. Let $z = \sqrt{a^2 + f(a)^2}$. By spherical symmetry, it can be seen that the above set is of measure at most $(1-z^2)^{\frac{d-1}{2}}$, and of measure greater or equal to the measure of $\Gamma = \{(u_1, \ldots, u_d) \in S^{d-1} \mid u_1 \geq z \text{ and } 0 \leq u_2 \leq c(u_1 - z)\}$, where $1 \geq c \geq \tan(\frac{\arccos(a)}{2})$ (for any $a \in [0, 1/2]$, c is in the range $(1/2, 1)$). Hence, it suffices to bound the measure of Γ by below.

Let $\delta = c\sqrt{\frac{\log(d)}{d}}$ for a sufficiently large constant c. The set Γ above is of measure larger than the measure of the δ neighborhood of

$$N = \left\{(u_1, \ldots, u_d) \in S^{d-1} \mid u_1 = z + \delta\left(1 + \frac{2}{c}\right) \text{ and } u_2 = \delta\right\}$$

for c as above. (The δ neighborhood of N is defined as in Claim 9.7.) We denote this set as N_δ. By Claim 9.7, the measure of N_δ is at least of value $(1 - z^2 - \theta(\delta))^{\frac{d-1}{2}}$. (Notice that $z > 1/k$, which in turn is independent of d; thus Claim 9.7 can be applied in our case.) Hence, the measure of $N(v) \cap \mathcal{C}_a$ is at least $(1 - \theta(\delta))^{\frac{d-1}{2}}(1 - z^2)^{\frac{d-1}{2}}$, which concludes our proof. ∎

THEOREM 3.8 (restated). *Let $a = (\frac{\log(d)}{d})^{\frac{1}{4}}$. Let k satisfy $\frac{1}{k-1} = a^2$. Let $r \in S^{d-1}$, and let $\mathcal{C}_a$ be an a-cap centered at r. Let $\varepsilon(a)$ be the value of $E(\mathcal{C}_a, \mathcal{C}_a)$. The value of $\varepsilon(a)$ is in the range*

$$\left[\frac{1}{poly(d)}\left(1-\frac{2(k-1)}{k-2}a^2\right)^{\frac{d-1}{2}},\ \left(1-\frac{2(k-1)}{k-2}a^2\right)^{\frac{d-1}{2}}\right].$$

Proof. We start by stating the following claim.

CLAIM 9.8. *Let $\mathcal{C}_a$ be an a-cap centered at $r = (1, 0, \ldots, 0)$. Let v be the vertex $(a, \sqrt{1-a^2}, 0, \ldots, 0)$ on the boundary of $\mathcal{C}_a$. Let $N(v)$ be the set of neighbors of v. Let $z = \sqrt{a^2 + \frac{(1/(k-1)+a^2)^2}{1-a^2}}$. If $z^4 = O(\log(d)/d)$, then the measure of vertices in $N(v)$ which are in the cap $\mathcal{C}_a$ satisfies*

$$\frac{1}{poly(d)}\left(1-z^2\right)^{\frac{d-1}{2}} \le \mu(N(v)\cap\mathcal{C}_a) \le \left(1-z^2\right)^{\frac{d-1}{2}}.$$

Proof. The set $N(v)\cap\mathcal{C}_a$ is equal to

$$\left\{(u_1,\ldots,u_d)\in S^{d-1} \mid u_1 \ge a \text{ and } u_2 \le -\frac{1/(k-1)+au_1}{\sqrt{1-a^2}}\right\}.$$

Denote $\frac{1/(k-1)+a^2}{\sqrt{1-a^2}}$ by $f(a)$. Let $z = \sqrt{a^2+f(a)^2}$. By spherical symmetry, it can be seen that the above set is of measure at most $(1-z^2)^{\frac{d-1}{2}}$, and of measure greater than or equal to the measure of $\Gamma = \{(u_1,\ldots,u_d)\in S^{d-1} \mid u_1 \ge z \text{ and } 0 \le u_2 \le c(u_1 - z)\}$, where $1 \ge c \ge \tan(\frac{\arccos(a)}{2})$. Hence, it suffices to bound the measure of Γ by below.

Let $r_1,\ldots,r_d$ be independent standard normal variables, let $s = \sqrt{\sum_{i=1}^d r_i^2}$, and let $s_3 = \sqrt{\sum_{i=3}^d r_i^2}$. Let $\phi(x)$ be the standard normal density function. The measure of Γ is given by the probability

$$\begin{aligned}
\mu(\Gamma) &= \Pr\left[\frac{r_1}{s} \ge z \text{ and } 0 \le \frac{r_2}{s} \le c\left(\frac{r_1}{s} - z\right)\right] \\
&\ge \Pr\left[r_1 \ge \sqrt{\frac{z^2(r_2^2+s_3^2)}{1-z^2}} \text{ and } 0 \le r_2 \le A\right] \\
&\ge \Pr[s_3^2 \le d]\Pr\left[r_1 \ge \sqrt{\frac{z^2(r_2^2+d)}{1-z^2}} \text{ and } 0 \le r_2 \le A_d\right] \\
&\ge \frac{1}{4}\Pr\left[r_1 \ge \sqrt{\frac{z^2 d}{1-z^2}} \text{ and } 0 \le r_2 \le \min(A_d, B)\right] \\
&\ge \frac{1}{4}\Pr\left[\sqrt{\frac{z^2 d}{1-z^2}} \le r_1 \le \sqrt{\frac{z^2(d+1)}{1-z^2}} \text{ and } 0 \le r_2 \le \min(A_d, B)\right],
\end{aligned}$$

where $A = c\frac{r_1 - z\sqrt{r_1^2(1+c^2-c^2z^2)+s_3^2(1-c^2z^2)}}{1-c^2z^2}$, $A_d = c\frac{r_1 - z\sqrt{r_1^2(1+c^2-c^2z^2)+d(1-c^2z^2)}}{1-c^2z^2}$, and B is of value $\sqrt{\frac{r_1^2(1-z^2)}{z^2} - d}$. Notice that we have used the fact that $\Pr[s_3^2 \le d] \ge 1/4$.

We proceed by evaluating $\min(A_d, B)$. Both A_d and B are functions of r_1. Furthermore, r_1^2 is in the range $[\frac{z^2 d}{1-z^2}, \frac{z^2(d+1)}{1-z^2}]$. Letting $x \in [0,1]$, we can represent A_d and B as functions of x by setting r_1 to be $\sqrt{\frac{z^2(d+x)}{1-z^2}}$. That is, $A_d =$

$z\frac{\sqrt{d+x}-\sqrt{d+xz^2(1+c^2-c^2z^2)}}{\sqrt{1-z^2}(1-c^2z^2)}$ and $B=\sqrt{x}$. It is not hard to verify that for our value of z it is the case that $\min(A_d,B)=A_d$. Thus

$$\begin{aligned}
\mu(\Gamma) &\geq \frac{1}{4}\Pr\left[\sqrt{\frac{z^2d}{1-z^2}}\leq r_1\leq\sqrt{\frac{z^2(d+1)}{1-z^2}}\text{ and }0\leq r_2\leq A_d\right]\\
&= w\int_{r_1\in R_1}\int_{r_2=0}^{A_d} e^{-\frac{r_1^2+r_2^2}{2}} \geq w\int_{r_1\in R_1}\frac{e^{-\frac{r_1^2}{2}}}{A_d}\int_{r_2=0}^{A_d} r_2e^{-\frac{r_2^2}{2}}\\
&= w\int_{r_1\in R_1}\frac{e^{-\frac{r_1^2}{2}}}{A_d}(1-e^{-\frac{A_d^2}{2}}) \geq w\int_{r_1\in R_1} A_de^{-\frac{r_1^2}{2}}\\
&\geq w\int_{x=0}^{1} A_de^{-\frac{z^2(d+x)}{2(1-z^2)}}\frac{z}{2\sqrt{1-z^2}\sqrt{d+x}}dx\\
&\geq w\frac{z^2}{d(1-z^2)(1-c^2z^2)}e^{-\frac{dz^2}{2(1-z^2)}}\int_{x=0}^{1} xe^{-\frac{z^2x}{2(1-z^2)}}\\
&\geq w\frac{z^2}{d(1-z^2)(1-c^2z^2)}e^{-\frac{dz^2}{2(1-z^2)}}\\
&\geq w\frac{z^2}{d(1-z^2)(1-c^2z^2)}\left(1-\frac{z^2}{1-z^2}\right)^{\frac{d-1}{2}}\\
&\geq w\frac{z^2}{d(1-z^2)(1-c^2z^2)}\left(1-z^2\right)^{\frac{d-1}{2}}\left(1-2z^4\right)^{\frac{d-1}{2}},
\end{aligned}$$

where R_1 is the range $[\sqrt{\frac{z^2d}{1-z^2}},\sqrt{\frac{z^2(d+1)}{1-z^2}}]$, and w is some constant independent of d (the value of w may change from line to line). Above we use the fact that $e^x>1+x$ and the fact that A_d is bounded by above by a constant independent of d. In the final inequalities we use a change of variables $r_1=\sqrt{\frac{z^2(d+x)}{1-z^2}}$ and the fact that $1+\alpha/3\leq\sqrt{1+\alpha}\leq 1+\alpha/2$ for $\alpha<3$. As $\mu(\Gamma)\geq\frac{1}{poly(d)}(1-z^2)^{\frac{d-1}{2}}$ for $z^4=O(\log(d)/d)$, we conclude our assertion. □

For the proof of Theorem 3.8, let $z=\sqrt{a^2+\frac{(1/(k-1)+a^2)^2}{1-a^2}}$ (as in Claim 9.8); thus $z^4=O(\log(d)/d)$. For the upper bound, use a bound on the measure of $\mathcal{C}_a$ and the upper bound in Claim 9.8. As for the lower bound, let $\delta=2(\frac{\log(d)}{d})^{\frac{3}{4}}$. Such a choice of δ will satisfy $a\delta\geq 2\frac{\log(d)}{d}$, and $z\delta=O(\log(d)/d)$.

Let $w=(w_1,w_2,\ldots,w_d)\in\mathcal{C}_a$ with first coordinate w_1 of value $a+\delta$. Consider a vertex $v=(v_1,v_2,\ldots,v_d)\in\mathcal{C}_a$ with first coordinate v_1 of value less than $a+\delta$. By Claim 9.8, it is not hard to verify that the measure of $N(v)\cap\mathcal{C}_a$ is greater than the measure of $N(w)\cap\mathcal{C}_a$, which is greater than $\frac{1}{poly(d)}\left(1-cz\delta\right)^{\frac{d-1}{2}}\left(1-z^2\right)^{\frac{d-1}{2}}\geq\frac{1}{poly(d)}(1-z^2)^{\frac{d-1}{2}}$ for some constant c. (We use Claim 9.2 and the fact that $z\delta=O(\log(d)/d)$.) As $a\delta\geq 2\log(d)/d$, we conclude, using Corollary 9.3, that the measure of edges in $E(\mathcal{C}_a,\mathcal{C}_a)$ is at least

$$\begin{aligned}
(\rho(a)-\rho(a+\delta))\frac{1}{poly(d)}\left(1-z^2\right)^{\frac{d-1}{2}} &\geq \frac{\rho(a)}{poly(d)}\left(1-z^2\right)^{\frac{d-1}{2}}\\
&\geq \frac{1}{poly(d)}\left(1-a^2\right)^{\frac{d-1}{2}}\left(1-z^2\right)^{\frac{d-1}{2}}.
\end{aligned}$$

Simplifying the above expression with the help of Claim 9.2, we then conclude our assertion. □

Acknowledgment. We would like to thank Luca Trevisan for his suggestion to analyze *edge* sampling.

REFERENCES

[AK98] N. Alon and N. Kahale, *Approximating the independence number via the θ function*, Math. Program., 80 (1998), pp. 253–264.

[AK02] N. Alon and M. Krivelevich, *Testing k-colorability*, SIAM J. Discrete Math., 15 (2002), pp. 211–227.

[Ben84] Y. Benyamini, *Two point symmetrization, the isoperimetric inequality on the sphere and some applications*, Longhorn Notes, University of Texas, Texas Funct. Anal. Seminar, 1983–1984, pp. 53–76.

[Bez02] S. Bezrukov, *Edge-isoperimetric problems of graphs*, in Graph Theory and Combinatorial Biology, Bolyai Soc. Math. Stud. 7, L. Lovasz, A. Gyarfas, G. O. H. Katona, A. Recski, and L. Szekely, eds., Budapest, 1999, pp. 157–197.

[BK97] A. Blum and D. Karger, *An $o(n^{3/14})$-coloring for 3-colorable graphs*, Inform. Process. Lett., 61 (1997), pp. 49–53.

[BT76] A. Baernstein and A. Taylor, *Spherical rearrangements, subharmonic functions, and *-functions in n-space*, Duke Math J., 43 (1976), pp. 245–268.

[Cha02] M. Charikar, *On semidefinite programming relaxations for graph coloring and vertex cover*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2002, pp. 616–620.

[Fei97] U. Feige, *Randomized graph products, chromatic numbers, and the Lovász theta function*, Combinatorica, 17 (1997), pp. 79–90.

[Fei02] U. Feige, *Approximating Maximum Clique by Removing Subgraphs*, manuscript, 2002.

[FK98] U. Feige and J. Kilian, *Zero knowledge and the chromatic number*, J. Comput. System Sci., 57 (1998), pp. 187–199.

[FS02] U. Feige and G. Schechtman, *On the optimality of the random hyperplane rounding technique for MAX CUT*, Random Structures Algorithms, 20 (2002), pp. 403–440.

[Fel66] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 2, John Wiley & Sons, New York, 1966.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.

[Hal93] M. Halldorsson, *A still better performance guarantee for approximate graph coloring*, Inform. Process. Lett., 45 (1993), pp. 19–23.

[Hås99] J. Håstad, *Clique is hard to approximate within $n^{1-\varepsilon}$*, Acta Math., 182 (1999), pp. 105–142.

[HNZ01] E. Halperin, R. Nathaniel, and U. Zwick, *Coloring k-colorable graphs using smaller palettes*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, pp. 319–326.

[KKL88] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on Boolean functions*, in Proceedings of 29th IEEE Symposium on the Foundations of Computer Science, White Plains, NY, 1988, pp. 68–80.

[KMS98] D. Karger, R. Motwani, and M. Sudan, *Approximate graph coloring by semidefinite programming*, J. ACM, 45 (1998), pp. 246–265.

[Kho01] S. Khot, *Improved inapproximability results for max clique, chromatic number and approximate graph coloring*, in Proceedings of 42nd IEEE Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, pp. 600–609.

[GK98] J. Kleinberg and M. X. Goemans, *The Lovász theta function and a semidefinite programming relaxation of vertex cover*, SIAM J. Discrete Math., 11 (1998), pp. 196–204.

[Lov79] L. Lovász, *On the Shannon capacity of a graph*, IEEE Trans. Inform. Theory, 25 (1979), pp. 2–13.

[Sze94] M. Szegedy, *A note on the θ number of Lovász and the generalized Delsarte bound*, in Proceedings of 35th IEEE Symposium on Foundations of Computer Science, Santa Fe, NM, 1994, pp. 36–39.

SIAM J. COMPUT.
Vol. 33, No. 6, pp. 1369–1416

DISJOINT NP-PAIRS*

CHRISTIAN GLASSER†, ALAN L. SELMAN‡, SAMIK SENGUPTA‡, AND LIYU ZHANG‡

Abstract. We study the question of whether the class DisjNP of disjoint pairs (A, B) of NP-sets contains a complete pair. The question relates to the question of whether optimal proof systems exist, and we relate it to the previously studied question of whether there exists a disjoint pair of NP-sets that is NP-hard. We show under reasonable hypotheses that nonsymmetric disjoint NP-pairs exist, which provides additional evidence for the existence of P-inseparable disjoint NP-pairs.

We construct an oracle relative to which the class of disjoint NP-pairs does not have a complete pair; an oracle relative to which optimal proof systems exist, and hence complete pairs exist, but no pair is NP-hard; and an oracle relative to which complete pairs exist, but optimal proof systems do not exist.

Key words. disjoint NP-pairs, promise problems, propositional proof systems, oracles, symmetry

AMS subject classification. 68Q15

DOI. 10.1137/S0097539703425848

1. Introduction.

We study the class DisjNP of disjoint pairs (A, B), where A and B are nonempty, disjoint sets belonging to NP. Such disjoint NP-pairs are interesting for at least two reasons. First, Grollmann and Selman [GS88] showed that the question of whether DisjNP contains P-inseparable disjoint NP-pairs is related to the existence of public-key cryptosystems. Second, Razborov [Raz94] and Pudlák [Pud03] demonstrated that these pairs are closely related to the theory of proof systems for propositional calculus. Specifically, Razborov showed that existence of an optimal propositional proof system implies existence of a complete pair for DisjNP. Primarily in this paper we are interested in the question raised by Razborov [Raz94] of whether DisjNP contains a complete pair. We show connections between this question and earlier work on disjoint NP-pairs, and we exhibit an oracle relative to which DisjNP does not contain any complete pair.

From a technical point of view, disjoint pairs are simply an equivalent formulation of promise problems. There are natural notions of reducibilities between promise problems [ESY84, Sel88] that disjoint pairs inherit easily [GS88]. Hence, completeness and hardness notions follow naturally. We begin in the next section with these definitions, some easy observations, and a review of the known results.

In section 3 we observe that if DisjNP does not contain a Turing-complete disjoint NP-pair, then DisjNP does not contain a disjoint NP-pair all of whose separators are Turing-hard for NP. The latter is a conjecture formulated by Even, Selman, and Yacobi [ESY84] and has several known consequences: Public-key cryptosystems that are NP-hard to crack do not exist; NP $\neq$ UP, NP $\neq$ coNP, and NPMV $\not\subseteq_c$

*Received by the editors April 16, 2003; accepted for publication (in revised form) May 12, 2004; published electronically August 27, 2004.

http://www.siam.org/journals/sicomp/33-6/42584.html

†Lehrstuhl für Informatik IV, Universität Würzburg, Am Hubland, 97074 Würzburg, Germany (glasser@informatik.uni-wuerzburg.de). The research of this author was performed at the University at Buffalo with support by a postdoctoral grant from the German Academic Exchange Service (Deutscher Akademischer Austauschdienst—DAAD).

‡Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY 14260 (selman@cse.buffalo.edu, samik@cse.buffalo.edu, lzhang7@cse.buffalo.edu). The research of the second author was partially supported by NSF grant CCR-0307077.

NPSV. Our main result in this section is an oracle X relative to which DisjNP does not contain a Turing-complete disjoint NP-pair and relative to which P $\neq$ UP. Relative to X, by Razborov's result [Raz94], optimal propositional proof systems do not exist. P-inseparable disjoint NP-pairs exist relative to X, because P $\neq$ UP [GS88]. Most researchers believe that P-inseparable disjoint NP-pairs exist, and we believe that no disjoint NP-pair has only NP-hard separators. Both of these properties hold relative to X. This is the first oracle relative to which both of these conditions hold simultaneously. Homer and Selman [HS92] obtained an oracle relative to which all disjoint NP-pairs are P-separable, so the conjecture of Even, Selman, and Yacobi holds relative to their oracle only for this trivial reason. Now let us say a few things about the construction of oracle X. Previous researchers have obtained oracles relative to which certain (promise) complexity classes do not have complete sets. However, the technique of Gurevich [Gur83], who proved that NP$\cap$coNP has Turing-complete sets if and only if it has many-one-complete sets, does not apply. Neither does the technique of Hemaspaandra, Jain, and Vereshchagin [HJV93], who demonstrated, among other results, an oracle relative to which FewP does not have a Turing-complete set.

In section 4 we show that the question of whether DisjNP contains a Turing-complete disjoint NP-pair has an equivalent natural formulation as a hypothesis about classes of single-valued partial functions. Section 5 studies *symmetric* disjoint NP-pairs. Pudlák [Pud03] defined a disjoint pair (A, B) to be symmetric if (A, B) is many-one reducible to (B, A). P-separable easily implies symmetric. We give complexity-theoretic evidence of the existence of nonsymmetric disjoint NP-pairs. As a consequence, we obtain new ways to demonstrate existence of P-inseparable sets. Also, we use symmetry to show under reasonable hypotheses that many-one and Turing reducibilities differ for disjoint NP-pairs. (All reductions in this paper are polynomial-time-bounded.) Concrete candidates for P-inseparable disjoint NP-pairs come from problems in UP or in NP $\cap$ coNP. Nevertheless, Grollmann and Selman [GS88] proved that the existence of P-inseparable disjoint NP-pairs implies the existence of P-inseparable disjoint NP-pairs, where both sets are NP-complete. Here we prove two analogous results. Existence of nonsymmetric disjoint NP-pairs implies existence of nonsymmetric disjoint NP-pairs, where both sets are NP-complete. If there exists a many-one-complete disjoint NP-pair, then there exists such a pair where both sets are NP-complete. Natural candidates for nonsymmetric or $\leq_m^{pp}$-complete disjoint NP-pairs arise either from cryptography or from proof systems [Pud03]. Our theorems show that the existence of such pairs will imply that nonsymmetric (or $\leq_m^{pp}$-complete) disjoint NP-pairs exist where both sets of the pair are $\leq_m^p$-complete for NP.

Section 6 constructs two oracles O_1 and O_2 that possess several interesting properties. First, let us mention some properties that hold relative to both of these oracles. Relative to both oracles, many-one-complete disjoint NP-pairs exist. Therefore, while we expect that complete disjoint NP-pairs do not exist, this is not provable by relativizable techniques. P-inseparable disjoint NP-pairs exist relative to these oracles, which we obtain by proving that nonsymmetric disjoint NP-pairs exist. The conjecture of Even, Selman, and Yacobi holds. Therefore, while nonexistence of Turing-complete disjoint NP-pairs is a sufficient condition for this conjecture, the converse does not hold, even in worlds in which P-inseparable pairs exist. Also, relative to these oracles, there exist P-inseparable pairs that are symmetric. Whereas nonsymmetric implies P-inseparability, again, we see that the converse does not hold.

In section 6 we discuss the properties of these oracles in detail. Relative to O_1, optimal proof systems exist, while relative to O_2, optimal proof systems do not exist. In particular, relative to O_2, the converse of Razborov's result does not hold. (That

is, relative to O_2, many-one complete pairs exist.)

The construction of O_2 involves some aspects that are unusual in complexity theory. We introduce undecidable requirements, and as a consequence, the oracle is undecidable. In particular, we need to define sets A and B, such that relative to O_2, the pair (A, B) is many-one complete. Therefore, we need to show that for every two nondeterministic, polynomial-time-bounded oracle Turing machines NM_i and NM_j, either $L(NM_i^{O_2})$ and $L(NM_j^{O_2})$ are not disjoint or there is a reduction from the disjoint pair $(L(NM_i^{O_2}), L(NM_j^{O_2}))$ to (A, B). We accomplish this as follows: Given NM_i, NM_j, and a finite initial segment X of O_2, we prove that either there is a finite extension Y of X such that for all oracles Z that extend Y,

$$L(NM_i^Z) \cap L(NM_j^Z) \neq \emptyset$$

or there is a finite extension Y of X such that for all oracles Z that extend Y,

$$L(NM_i^Z) \cap L(NM_j^Z) = \emptyset.$$

Then we select the extension Y that exists. In this manner we *force* one of these two conditions to hold.

In the latter case, to obtain a reduction from the pair $(L(NM_i^{O_2}), L(NM_j^{O_2}))$ to (A, B) requires encoding information into the oracle O_2. The other conditions that we want O_2 to satisfy require diagonalizations. In order to prove that there is room to diagonalize, we need to carefully control the number of words that must be reserved for encoding. This is a typical concern in oracle constructions, but even more so here. We manage this part of the construction by inventing a unique data structure that stores words reserved for the encoding, and then prove that we do not store too many such words.

2. Preliminaries. We fix the alphabet $\Sigma = \{0, 1\}$, and we denote the length of a word w by $|w|$. The set of all (resp., nonempty) words is denoted by Σ^* (resp., Σ^+). Let $\Sigma^{<n} \stackrel{df}{=} \{w \in \Sigma^* \mid |w| < n\}$, and define $\Sigma^{\leq n}$, $\Sigma^{\geq n}$, and $\Sigma^{>n}$ analogously. For a set of words X let $X^{<n} \stackrel{df}{=} X \cap \Sigma^{<n}$, and define $X^{\leq n}$, $X^{=n}$, $X^{\geq n}$, and $X^{>n}$ analogously. For sets of words we take the complement with respect to Σ^*. For $A, B \subseteq \Sigma^*$ let $A \oplus B \stackrel{df}{=} \{0x \mid x \in A\} \cup \{1y \mid y \in B\}$.

The set of (nonzero) natural numbers is denoted by $\mathbb{N}$ (resp., $\mathbb{N}^+$). We use polynomial-time computable and polynomial-time invertible pairing functions $\langle \cdot, \cdot \rangle : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ and $\langle \cdot, \cdot, \cdot \rangle : \mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$. For a function f, $\mathrm{dom}(f)$ denotes the domain of f.

Cook and Reckhow [CR79] defined a *propositional proof system* (proof system, for short) to be a function $f : \Sigma^* \to \mathrm{TAUT}$ such that f is onto and $f \in \mathrm{PF}$. (TAUT denotes the set of tautologies.) Note that f is not necessarily honest; it is possible that a formula $\phi \in \mathrm{TAUT}$ has only exponentially long proofs w, i.e., $f(w) = \phi$ and $|w| = 2^{\Omega(|\phi|)}$.

Let f and f' be two proof systems. We say that f *simulates* f' if there is a polynomial p and a function $h : \Sigma^* \to \Sigma^*$ such that for every $w \in \Sigma^*$, $f(h(w)) = f'(w)$ and $|h(w)| \leq p(|w|)$. If, additionally, $h \in \mathrm{PF}$, then we say that f *p-simulates* f'.

A proof system is *optimal* (resp., *p-optimal*) if it simulates (resp., p-simulates) every other proof system. The notion of simulation between proof systems is analogous to the notion of reducibility between problems. Using that analogy, optimal proof systems correspond to complete problems.

2.1. Disjoint pairs, separators, and a conjecture. We begin with the following definition.

DEFINITION 2.1. *A* disjoint NP-pair *(*NP*-pair, for short) is a pair of nonempty sets A and B such that $A, B \in$ NP and $A \cap B = \emptyset$. Let* DisjNP *denote the class of all disjoint* NP*-pairs.*

Given a disjoint NP-pair (A, B), a *separator* is a set S such that $A \subseteq S$ and $B \subseteq \overline{S}$; we say that S *separates* (A, B). Let $Sep(A, B)$ denote the class of all separators of (A, B). For disjoint NP-pairs (A, B), the fundamental question is whether $Sep(A, B)$ contains a set belonging to P. In that case the pair is P*-separable*; otherwise, the pair is P*-inseparable*. The following proposition summarizes the known results about P-separability.

PROPOSITION 2.2.

1. P $\neq$ NP $\cap$ coNP *implies that* NP *contains* P*-inseparable sets.*
2. P $\neq$ UP *implies that* NP *contains* P*-inseparable sets* [GS88].
3. *If* NP *contains* P*-inseparable sets, then* NP *contains* NP*-complete* P*-inseparable sets* [GS88].

While it is probably the case that NP contains P-inseparable sets, there is an oracle relative to which P $\neq$ NP and P-inseparable sets in NP do not exist [HS92]. So P $\neq$ NP probably is not a sufficiently strong hypothesis to show existence of P-inseparable sets in NP.

DEFINITION 2.3. *Let (A, B) be a disjoint* NP*-pair.*

1. $X \leq_m^{pp} (A, B)$ *if, for every separator S of (A, B), $X \leq_m^p S$.*
2. $X \leq_T^{pp} (A, B)$ *if, for every separator S of (A, B), $X \leq_T^p S$.*
3. (A, B) *is* NP-hard *if* SAT $\leq_T^{pp} (A, B)$.
4. (A, B) *is* uniformly NP-hard *if there is a deterministic polynomial-time oracle Turing machine M such that for every $S \in Sep(A, B)$,* SAT $\leq_T^p S$ *via M.*

Grollmann and Selman [GS88] showed that NP-hard implies uniformly NP-hard; i.e., both statements of the definition are equivalent. Even, Selman, and Yacobi [ESY84] conjectured that there does not exist a disjoint NP-pair (A, B) such that all separators of (A, B) are $\leq_T^p$ hard for NP.

CONJECTURE 2.4 (see [ESY84]). *There do not exist disjoint* NP*-pairs that are* NP*-hard.*

If Conjecture 2.4 holds, then no public-key cryptosystem is NP-hard to crack [ESY84]. This conjecture is a strong hypothesis with the following known consequences. In section 3 we show a sufficient condition for Conjecture 2.4 to hold.

PROPOSITION 2.5 (see [ESY84, GS88, Sel94]). *If Conjecture* 2.4 *holds, then* NP $\neq$ coNP*,* NP $\neq$ UP*, and* NPMV $\not\subseteq_c$ NPSV*.*

2.2. Reductions for disjoint pairs. We review the natural notions of reducibilities between disjoint pairs [GS88].

DEFINITION 2.6 (nonuniform reductions for pairs). *Let (A, B) and (C, D) be disjoint pairs.*

1. (A, B) *is* many-one reducible in polynomial-time *to (C, D), $(A, B) \leq_m^{pp} (C, D)$, if for every separator $T \in Sep(C, D)$ there exists a separator $S \in Sep(A, B)$ such that $S \leq_m^p T$.*
2. (A, B) *is* Turing reducible in polynomial-time *to (C, D), $(A, B) \leq_T^{pp} (C, D)$, if for every separator $T \in Sep(C, D)$ there exists a separator $S \in Sep(A, B)$ such that $S \leq_T^p T$.*

DEFINITION 2.7 (uniform reductions for pairs). *Let (A, B) and (C, D) be disjoint pairs.*

1. (A, B) *is* uniformly many-one reducible in polynomial-time *to* (C, D), $(A, B) \leq^{pp}_{um} (C, D)$, *if there exists a polynomial-time computable function* f *such that for every separator* $T \in Sep(C, D)$, *there exists a separator* $S \in Sep(A, B)$ *such that* $S \leq^{p}_{m} T$ *via* f.
2. (A, B) *is* uniformly Turing reducible in polynomial-time *to* (C, D), $(A, B) \leq^{pp}_{uT} (C, D)$, *if there exists a polynomial-time oracle Turing machine* M *such that for every separator* $T \in Sep(C, D)$, *there exists a separator* $S \in Sep(A, B)$ *such that* $S \leq^{p}_{T} T$ *via* M.

If f and M are as above, then we also say that $(A, B) \leq^{pp}_{um} (C, D)$ via f and $(A, B) \leq^{pp}_{uT} (C, D)$ via M. Observe that if $(A, B) \leq^{pp}_{m} (C, D)$ and (C, D) is P-separable, then so is (A, B) (and the same holds for $\leq^{pp}_{T}$, $\leq^{pp}_{um}$, and $\leq^{pp}_{uT}$). We retain the promise problem notation in order to distinguish from reducibilities between sets. Grollmann and Selman proved that Turing reductions and uniform Turing reductions are equivalent.

PROPOSITION 2.8 (see [GS88]). $(A, B) \leq^{pp}_{T} (C, D) \Leftrightarrow (A, B) \leq^{pp}_{uT} (C, D)$ *for all disjoint pairs* (A, B) *and* (C, D).

In order to obtain the corresponding theorem for $\leq^{pp}_{um}$, we can adapt the proof of Proposition 2.8, but a separate argument is required.

LEMMA 2.9. *Let* S *and* T *be nonempty, disjoint sets. Let* X *and* Y *be nonempty, finite, disjoint sets such that* $X \cap T = \emptyset$ *and* $Y \cap S = \emptyset$. *Then the disjoint pairs* (S, T) *and* $(S \cup X, T \cup Y)$ *are equivalent by polynomial-time uniform reductions.*

Proof. First we show that $(S \cup X, T \cup Y) \leq^{pp}_{um} (S, T)$. Choose $a \in S$ and $b \in T$. Define the polynomial-time computable function f by

$$f(x) \stackrel{df}{=} \begin{cases} a & \text{if } x \in X, \\ b & \text{if } x \in Y, \\ x & \text{otherwise.} \end{cases}$$

Let $A \in Sep(S, T)$. We need to see that $f^{-1}(A) \in Sep(S \cup X, T \cup Y)$. So we show that

1. $S \cup X \subseteq f^{-1}(A)$, and
2. $T \cup Y \subseteq \overline{f^{-1}(A)}$.

For item 1, if $x \in X$, then $f(x) = a \in S \subseteq A$. So $f(X) \subseteq A$. Hence, $X \subseteq f^{-1}(A)$. If $x \in S - X$, then $f(x) = x \in S \subseteq A$. So, $S - X \subseteq f^{-1}(A)$. For item 2, if $x \in Y$, then $f(x) = b \in T \subseteq \overline{A}$. So $f(Y) \cap A = \emptyset$. That is, $Y \subseteq \overline{f^{-1}(A)}$. If $x \in T - Y$, then $f(x) = x \in T$. So $f(T - Y) \cap A = \emptyset$. That is, $T - Y \subseteq \overline{f^{-1}(A)}$.

Every separator of $(S \cup X, T \cup Y)$ is a separator of (S, T). Therefore, the identity function provides a uniform reduction from (S, T) to $(S \cup X, T \cup Y)$. □

THEOREM 2.10. $\leq^{pp}_{m} = \leq^{pp}_{um}$.

Proof. Assume that (Q, R) is not uniformly many-one reducible to (S, T). That is, for every polynomial-time computable function f, there exists a set $A \in Sep(S, T)$ such that $f^{-1}(A) \notin Sep(Q, R)$. Then for every polynomial-time computable function f, there exists $A \in Sep(S, T)$ and a string y that witnesses the fact that $f^{-1}(A) \notin Sep(Q, R)$. Namely, either

$$y \in Q \wedge y \notin f^{-1}(A) \text{ (i.e., } f(y) \notin A) \qquad \text{or} \qquad y \in R \wedge y \in f^{-1}(A) \text{ (i.e., } f(y) \in A).$$

We will show from this assumption that (Q, R) is not many-one reducible to (S, T). We will construct a decidable separator A of (S, T) such that for every polynomial-time computable function f, $f^{-1}(A)$ is not a separator of (Q, R). Let $\{f_i\}_{i \geq 1}$ be an

effective enumeration of the polynomial-time computable functions with associated polynomial-time bounds $\{p_i\}_{i\geq 1}$.

The separator A of (S,T) will be constructed inductively to be of the form $S \cup \bigcup\{Y_i \mid i \geq 1\}$, where $\bigcup\{Y_i \mid i \geq 1\}$ is a subset of $\overline{T}$ and $Y_0 \subseteq Y_1 \subseteq \cdots$. At stage i of the construction, we will choose a finite subset Y_i of $\overline{T}$ such that $f^{-1}(S \cup Y_i)$ is not a separator of (Q,R).

Stage 0. Define $Y_0 = \{0\}$ and $n_0 = 1$.

Stage i ($i \geq 1$). By induction hypothesis, Y_{i-1} is defined, $n_{i-1} \geq 0$ is defined, and $Y_{i-1} \subseteq \overline{T} \cap \Sigma^{\leq n_{i-1}}$.

Now we state a sequence of claims.

CLAIM 2.11. *There exists a set* X, $X \subseteq \overline{T \cup \Sigma^{\leq n_{i-1}}}$, *and a witness* y_i *demonstrating that* $f_i^{-1}(S \cup Y_{i-1} \cup X)$ *is not a separator of* (Q,R). *That is,*

$$y_i \in Q \wedge y_i \notin f_i^{-1}(S \cup Y_{i-1} \cup X) \text{ (i.e., } f_i(y_i) \notin S \cup Y_{i-1} \cup X)$$

or

$$y_i \in R \wedge y_i \in f_i^{-1}(S \cup Y_{i-1} \cup X) \text{ (i.e., } f_i(y_i) \in S \cup Y_{i-1} \cup X).$$

If the claim is false, then for every $X \subseteq \overline{T \cup \Sigma^{\leq n_{i-1}}}$, $Q \subseteq f_i^{-1}(S \cup Y_{i-1} \cup X)$ and $R \subseteq \overline{f_i^{-1}(S \cup Y_{i-1} \cup X)}$. The set of all languages $S \cup Y_{i-1} \cup X$, where $X \subseteq \overline{T \cup \Sigma^{\leq n_{i-1}}}$, is exactly the set of separators of the disjoint pair

$$(S \cup Y_{i-1}, T \cup (\Sigma^{\leq n_{i-1}} - (S \cup Y_{i-1}))).$$

Thus, if the claim is false, then (Q,R) is uniformly many-one reducible to $(S \cup Y_{i-1}, T \cup (\Sigma^{\leq n_{i-1}} - Y_{i-1}))$. However, by Lemma 2.9, this contradicts the assumption that (Q,R) is not uniformly reducible to (S,T). Hence the claim is true.

CLAIM 2.12. *There exists a finite set* X, $X \subseteq \overline{T \cup \Sigma^{\leq n_{i-1}}}$, *and a witness* y_i *that satisfy the condition of Claim* 2.11.

For X and witness y_i, whose existence Claim 2.11 guarantees, $|f_i(y_i)| \leq p_i(|y_i|)$. So $X' = X \cap \Sigma^{\leq p_i(|y_i|)}$ and y_i satisfy the condition as well.

CLAIM 2.13. *There is an effective procedure that on input* (i, Y_{i-1}, n_{i-1}) *finds a finite set* $X \subseteq \overline{T \cup \Sigma^{\leq n_{i-1}}}$ *and witness* y_i *to satisfy the condition of Claim* 2.11.

This is trivial. Effectively enumerate pairs of finite sets and strings until a pair with the desired property is found.

At Stage i, apply Claim 2.13; define $Y_i = Y_{i-1} \cup X$ and define $n_i = 1 + \max(2^{n_{i-1}}, p_i(|y_i|))$.

Define $A = S \cup \bigcup\{Y_i \mid i \geq 1\}$. Since $\bigcup\{Y_i \mid i \geq 1\} \subseteq \overline{T}$, A is a separator of (S,T). It is easy to see that A is decidable. Finally, for every f_i, $i \geq 1$, $f_i^{-1}(A)$ is not a separator of (Q,R): Clearly this holds for $f_i^{-1}(S \cup Y_i)$, and the construction preserves this property. □

We obtain the following useful characterization of many-one reductions. Observe that this is the way Razborov [Raz94] defined reductions between disjoint pairs.

THEOREM 2.14. $(Q,R) \leq_m^{pp} (S,T)$ *if and only if there exists a polynomial-time computable function* f *such that* $f(Q) \subseteq S$ *and* $f(R) \subseteq T$.

Proof. By Theorem 2.10 there is a polynomial-time computable function f such for every $A \in Sep(S,T)$, $f^{-1}(A) \in Sep(Q,R)$. That is, if $A \in Sep(S,T)$, then $Q \subseteq f^{-1}(A)$ and $R \subseteq \overline{f^{-1}(A)}$, which implies that $f(Q) \subseteq A$ and $f(R) \cap A = \emptyset$. Well, $S \in Sep(S,T)$. So $f(Q) \subseteq S$. Also, $\overline{T} \in Sep(S,T)$. So $f(R) \cap \overline{T} = \emptyset$. That is, $f(R) \subseteq T$. The converse is immediate. □

3. Complete disjoint NP-pairs. Keeping with common terminology, a disjoint pair (A, B) is $\leq_m^{pp}$-complete ($\leq_T^{pp}$-complete) for the class DisjNP if $(A, B) \in$ DisjNP and for every disjoint pair $(C, D) \in$ DisjNP, $(C, D) \leq_m^{pp} (A, B)$ (resp., $(C, D) \leq_T^{pp} (A, B)$).

Consider the following assertions:

1. DisjNP does not have a $\leq_T^{pp}$-complete disjoint pair.
2. DisjNP does not have a $\leq_m^{pp}$-complete disjoint pair.
3. DisjNP does not contain a disjoint pair all of whose separators are $\leq_T^p$-hard for NP (i.e., Conjecture 2.4 holds).
4. DisjNP does not contain a disjoint pair all of whose separators are $\leq_m^p$-hard for NP.

Assertions 1 and 2 are possible answers to the question raised by Razborov [Raz94] of whether DisjNP contains complete disjoint pairs. Assertion 3 is Conjecture 2.4. Assertion 4 is the analogue of this conjecture using many-one reducibility.

We can dispense with assertion 4 immediately, for it is equivalent to NP $\neq$ coNP.

PROPOSITION 3.1. NP $\neq$ coNP *if and only if* DisjNP *does not contain a disjoint pair all of whose separators are* $\leq_m^p$*-hard for* NP.

Proof. If NP = coNP, then $(\mathrm{SAT}, \overline{\mathrm{SAT}})$ is a disjoint pair in DisjNP all of whose separators are $\leq_m^p$-hard for NP.

To show the other direction, consider the disjoint pair $(A, B) \in$ DisjNP and assume that all of its separators are $\leq_m^p$-hard for NP. Since $\overline{B}$ is a separator of (A, B), SAT $\leq_m^p \overline{B}$. Therefore, $\overline{\mathrm{SAT}} \leq_m^p B$, implying that $\overline{\mathrm{SAT}} \in$ NP. Thus, NP = coNP. □

PROPOSITION 3.2. *Assertion* 1 *implies assertions* 2 *and* 3. *Assertion* 2 *implies assertion* 4. *Assertion* 3 *implies assertion* 4.

This proposition states, in part, that assertion 1 is so strong as to imply Conjecture 2.4.

Proof. It is trivial that assertion 1 implies assertion 2, and that assertion 3 implies assertion 4.

We prove that assertion 1 implies assertion 3. Assume assertion 3 is false and let $(A, B) \in$ DisjNP such that all separators are NP-hard. We claim that (A, B) is $\leq_T^{pp}$-complete for DisjNP. Let (C, D) belong to DisjNP. Let S be an arbitrary separator of (A, B). Note that S is NP-hard and $C \in$ NP. So $C \leq_T^p S$. Since C is a separator of (C, D), this demonstrates that $(C, D) \leq_T^{pp} (A, B)$.

Similarly, we prove that assertion 2 implies assertion 4. In this case, every separator S of (A, B) is $\leq_m^p$-hard for NP. So $C \leq_m^p S$. Therefore, $(C, D) \leq_m^{pp} (A, B)$. □

Homer and Selman [HS92] constructed an oracle relative to which P $\neq$ NP and every disjoint NP-pair is P-separable. Relative to this oracle, assertion 3 holds and assertions 1 and 2 are false. To see this, let (A, B) be an arbitrary disjoint NP-pair. We show that (A, B) is both $\leq_T^{pp}$-complete and $\leq_m^{pp}$-complete. For any other pair $(C, D) \in$ DisjNP, since (C, D) is P-separable, there is a separator S of (C, D) that is in P. Therefore, for any separator L of (A, B), S trivially $\leq_m^p$-reduces and $\leq_T^p$-reduces to L. So $(C, D) \leq_m^{pp} (A, B)$ and $(C, D) \leq_T^{pp} (A, B)$.

There exists an oracle relative to which UP = NP $\neq$ coNP [GW03]. So, relative to this oracle assertion 4 holds, but assertion 3 is false. In section 6 we will construct oracles relative to which assertion 4 holds while assertions 1 and 2 fail.

In Theorem 3.8 we construct an oracle X relative to which assertion 1 is true. In Corollary 3.11 we observe that P $\neq$ UP relative to X. Therefore, by Proposition 3.2, all of the following properties hold relative to X:

1. DisjNP does not have a $\leq_T^{pp}$-complete disjoint pair.
2. Conjecture 2.4 holds; so UP $\neq$ NP, NP $\neq$ coNP, NPMV $\not\subseteq_c$ NPSV, and NP-hard public-key cryptosystems do not exist [ESY84, Sel94].
3. P $\neq$ UP; therefore P-inseparable disjoint NP-pairs exist [GS88].
4. Optimal propositional proof systems do not exist [Raz94].
5. There is a tally set $T \in \text{coNP} - \text{NP}$ and a tally set $T' \in \text{coNE} - \text{E}$ [Pud86, KP89].

The following lemma is essential to the proofs of Theorems 3.8 and 6.1. Intuitively this lemma says that, given two nondeterministic machines and some oracle, either we can force the languages accepted by these machines to be not disjoint, or we can ensure that one of the machines rejects a given string q by reserving only polynomially many strings.

LEMMA 3.3. *Let M and N be nondeterministic polynomial-time oracle Turing machines with polynomial-time bounds p_M and p_N, respectively. Let Y be an oracle and $q \in \Sigma^*$, $|q| = n$. Then, for any set T, at least one of the following holds:*

- *$\exists S \subseteq T$, $\|S\| \leq p_M(n) + p_N(n)$, such that $q \in L(M^{Y \cup S}) \cap L(N^{Y \cup S})$.*
- *$\exists S' \subseteq T$, $\|S'\| \leq p_M(n) \cdot (p_N(n) + 1)$, such that either* (i) *for any $S \subseteq T$, if $S \cap S' = \emptyset$, then $M^{Y \cup S}(q)$ rejects, or* (ii) *for any $S \subseteq T$, if $S \cap S' = \emptyset$, then $N^{Y \cup S}(q)$ rejects.*

Proof. Let us define the following languages:

- $L_M = \{\langle P, Q_y, Q_n\rangle \mid$ for some set $S_M \subseteq T$, P is an accepting path of $M^{Y \cup S_M}(q)$ and Q_y (resp., Q_n) is the set of words in S_M (resp., $T - (Y \cup S_M)$) that are queried on $P\}$.
- $L_N = \{\langle P, Q_y, Q_n\rangle \mid$ for some set $S_N \subseteq T$, P is an accepting path of $N^{Y \cup S_N}(q)$ and Q_y (resp., Q_n) is the set of words in S_N (resp., $T - (Y \cup S_N)$) that are queried on $P\}$.

We say that $\langle P, Q_y, Q_n\rangle \in L_M$ *conflicts* with $\langle P', Q'_y, Q'_n\rangle \in L_N$ if $Q_y \cap Q'_n \neq \emptyset$ or $Q'_y \cap Q_n \neq \emptyset$. In other words, there is a conflict if there exists at least one query that is in T and that is answered differently on P and P'.

Case I. There exist $\langle P, Q_y, Q_n\rangle \in L_M$ and $\langle P', Q'_y, Q'_n\rangle \in L_N$ that do not conflict.

Let $S = Q_y \cup Q'_y$. We claim in this case that $q \in L(M^{Y \cup S}) \cap L(N^{Y \cup S})$. Let S_M and S_N be the subsets of T that witness $\langle P, Q_y, Q_n\rangle \in L_M$ and $\langle P', Q'_y, Q'_n\rangle \in L_N$. So P is an accepting path of $M^{Y \cup S_M}(q)$, and P' is an accepting path of $N^{Y \cup S_N}(q)$. Assume that on P there exists a query r that is answered differently with respect to the oracles $Y \cup S_M$ and $Y \cup S$. Hence $r \notin Y$. Moreover, either $r \in S_M - S$ or $r \in S - S_M$. However, r cannot belong to $S_M - S$, since otherwise $r \in Q_y$, and therefore $r \in S$. So $r \in S - S_M$. Hence $r \notin Q_y$, and therefore $r \in Q'_y$. On the other hand, $r \in S - S_M$ implies $r \in T - (Y \cup S_M)$. Therefore, $r \in Q_n \cap Q'_y$, which contradicts the assumption in Case I. This shows that P is an accepting path of $M^{Y \cup S}(q)$. Analogously we show that P' is an accepting path of $N^{Y \cup S}(q)$. Hence $q \in L(M^{Y \cup S}) \cap L(N^{Y \cup S})$. Note that $\|S\| = \|Q_y \cup Q'_y\| \leq p_M(n) + p_N(n)$.

Case II. Every triple $\langle P, Q_y, Q_n\rangle \in L_M$ conflicts with every triple $\langle P', Q'_y, Q'_n\rangle \in L_N$.

Note that in this case we cannot have both a triple $\langle P, \emptyset, Q_n\rangle$ in L_M and a triple $\langle P', \emptyset, Q'_n\rangle$ in L_N, simply because these two triples do not conflict with each other. We use the following algorithm to create the set S' as claimed in the statement of this lemma.

$\mathtt{S'} = \emptyset$
`while` ($\mathtt{L_M} \neq \emptyset$ `and` $\mathtt{L_N} \neq \emptyset$)

```
(1)        Choose some (P*, Q*_y, Q*_n) ∈ L_M
(2)        S' = S' ∪ Q*_y ∪ Q*_n
(3)        For every t = (P, Q_y, Q_n) ∈ L_M
(4)           if Q_y ∩ (Q*_y ∪ Q*_n) ≠ ∅ then remove t
(5)        For every t' = (P', Q'_y, Q'_n) ∈ L_N
(6)           if Q'_y ∩ (Q*_y ∪ Q*_n) ≠ ∅ then remove t'
      end while
```

We claim that after k iterations of the *while* loop, for every triple $(P', Q'_y, Q'_n) \in L_N$, $\|Q'_n\| \geq k$. If this claim is true, the while loop iterates at most $p_N(n)+1$ times, since for any triple in L_N, $\|Q'_n\|$ is bounded by the running time of N on q, i.e., $p_N(n)$. On the other hand, during each iteration, S' is increased by at most $p_M(n)$ strings, since for any triple in L_M, $\|Q_y \cup Q_n\|$ is bounded by the running time of M on q, i.e., $p_M(n)$. Therefore, $\|S'\| \leq p_M(n) \cdot (p_N(n)+1)$ when this algorithm terminates.

CLAIM 3.4. *After the kth iteration of the* while *loop of the above algorithm, for every $t' = \langle P', Q'_y, Q'_n\rangle$ that remains in L_N, $\|Q'_n\| \geq k$.*

Proof. For every k, t_k denotes the triple $\langle P^k, Q^k_y, Q^k_n\rangle \in L_M$ that is chosen during the kth iteration in step (1). For every $t' = \langle P', Q'_y, Q'_n\rangle$ that is in L_N at the beginning of this iteration, t_k conflicts with t' (assumption of Case II). Therefore, there is a query in $(Q^k_n \cap Q'_y) \cup (Q^k_y \cap Q'_n)$. If this query is in $Q^k_n \cap Q'_y$, then t' will be removed from L_N in step (6). Otherwise, i.e., if $Q^k_y \cap Q'_n \neq \emptyset$, then let q' be the lexicographically smallest query in $Q^k_y \cap Q'_n$. In this case, t' will not be removed from L_N; we say that t' *survives* the kth iteration *due to query* q'. Note that t' can survive only due to a query that is in Q'_n. We will use this fact to prove that $\|Q'_n\| \geq k$ after k iterations.

We show now that any triple that is left in L_N after k iterations survives each iteration due to a different query. This will complete the proof of the claim. Assume that t' survives iteration k by query $q' \in Q^k_y \cap Q'_n$. If t' had survived an earlier iteration $l < k$ by the same query q', then q' is also in $Q^l_y \cap Q'_n$. Therefore, $Q^l_y \cap Q^k_y \neq \emptyset$. So $t_k = \langle P^k, Q^k_y, Q^k_n\rangle$ should have been removed in step (4) during iteration l, and cannot be chosen at the beginning of iteration k, as claimed. Hence, q' cannot be the query by which t' had survived iteration l. This proves Claim 3.4. □

Therefore, now we have a set $S' \subseteq T$ of the required size such that either L_M or L_N is empty. Assume that L_M is empty, and for some set $S_M \subseteq T$ it holds that $S_M \cap S' = \emptyset$ and $M^{(Y \cup S_M)}(q)$ accepts. Let P be an accepting path of $M^{(Y \cup S_M)}(q)$ and let Q_y (resp., Q_n) be the set of words in S_M (resp., $T - (Y \cup S_M)$) that are queried on P. The triple $\langle P, Q_y, Q_n\rangle$ must have been in L_M and has been removed during some iteration. This implies that during that iteration, $Q_y \cap S' \neq \emptyset$ (step (4)). Since $Q_y \subseteq S_M$, this contradicts the assumption that $S_M \cap S' = \emptyset$.

A similar argument holds for L_N. Hence either $L_M = \emptyset$ and $M^{(Y \cup S)}(q)$ rejects for any $S \subseteq T$ such that $S \cap S' = \emptyset$, or $L_N = \emptyset$ and $N^{(Y \cup S)}(q)$ rejects for any $S \subseteq T$ such that $S \cap S' = \emptyset$. This ends the proof of Lemma 3.3. □

We define the following notions for reductions relative to oracles.

DEFINITION 3.5. *For any set X, a pair of disjoint sets (A, B) is* polynomial-time Turing reducible relative to X *($\leq^{pp,X}_T$) to a pair of disjoint sets (C, D) if for any separator S that separates (C, D), there exists a polynomial-time deterministic oracle Turing machine M such that $M^{S \oplus X}$ accepts a language that separates (A, B).*

DEFINITION 3.6. *For any set X, let*

$$\text{DisjNP}^X = \{(A, B) \mid A \in \text{NP}^X,\ B \in \text{NP}^X,\ A \neq \emptyset,\ B \neq \emptyset,\ \textit{and}\ A \cap B = \emptyset\}.$$

(C, D) is $\leq^{pp,X}_T$-complete for DisjNP^X *if $(C, D) \in \text{DisjNP}^X$ and for all $(A, B) \in$*

DisjNP^X, $(A,B) \leq_T^{pp,X} (C,D)$. *Similarly,* (C,D) *is* $\leq_T^{pp}$*-complete for* DisjNP^X *if* $(C,D) \in \mathrm{DisjNP}^X$ *and for all* $(A,B) \in \mathrm{DisjNP}^X, (A,B) \leq_T^{pp} (C,D)$.

However, the following proposition shows that if there exists a disjoint pair that is Turing-complete relative to X, then there is a pair that is Turing-complete such that the reduction between the separators does not access the oracle.

PROPOSITION 3.7. *For any set* X, DisjNP^X *has a* $\leq_T^{pp,X}$*-complete pair if and only if* DisjNP^X *has a* $\leq_T^{pp}$*-complete pair.*

Proof. The *if* direction is trivial. We only show the *only if* direction. Suppose (C,D) is $\leq_T^{pp,X}$-complete for DisjNP^X. We claim that $(C \oplus X, D \oplus \overline{X})$ is $\leq_T^{pp}$-complete for DisjNP^X. Observe that $(C \oplus X, D \oplus \overline{X}) \in \mathrm{DisjNP}^X$. Consider any $(A,B) \in \mathrm{DisjNP}^X$. Let S' separate $(C \oplus X, D \oplus \overline{X})$. Define $S = \{x \mid 0x \in S'\}$. Then S separates (C,D) and $S' = S \oplus X$. Since (C,D) is $\leq_T^{pp,X}$-complete for DisjNP^X, there exists a polynomial-time oracle Turing machine M so that $L(M^{S \oplus X})$ separates (A,B). That is, $L(M^{S'})$ separates (A,B), which is what we needed to prove. □

THEOREM 3.8. *There exists an oracle* X *such that* DisjNP^X *does not have a* $\leq_T^{pp,X}$*-complete pair.*

Proof. By Proposition 3.7, it suffices to show that DisjNP^X has no $\leq_T^{pp}$-complete pair. By Proposition 2.8, it suffices to construct X such that for every $(C,D) \in \mathrm{DisjNP}^X$ there exists a disjoint pair $(A,B) \in \mathrm{DisjNP}^X$ such that $(A,B) \not\leq_{uT}^{pp} (C,D)$.

Suppose $\{M_k\}_{k \geq 1}$ (resp., $\{N_i\}_{i \geq 1}$) is an enumeration of deterministic (resp., nondeterministic) polynomial-time oracle Turing machines. Let r_k and p_i be the corresponding polynomial time bounds for M_k and N_i. For any r, s, d, let $\Sigma_{rs}^d = 0^r 10^s 1 \Sigma^d$ and $l_{rs}^d = r + s + d + 2$ (i.e., l_{rs}^d is the length of strings in Σ_{rs}^d). For $Z \subseteq \Sigma^*$, $i \geq 1$, and $j \geq 1$, define

$$A_{ij}^Z = \{0^n \mid \exists x,\ |x| = n,\ 0^i 10^j 10x \in Z\}$$

and

$$B_{ij}^Z = \{0^n \mid \exists x,\ |x| = n,\ 0^i 10^j 11x \in Z\}.$$

We construct the oracle in stages. X_m denotes the oracle before stage m. We define $X = \bigcup_{m \geq 1} X_m$. Initially, let $X = \emptyset$. In stage $m = \langle i,j,k \rangle$, we choose some number $n = n_m$ and add strings from Σ_{ij}^{n+1} to the oracle such that either $L(N_i^{X_{m+1}}) \cap L(N_j^{X_{m+1}}) \neq \emptyset$ or $(A_{ij}^{X_{m+1}}, B_{ij}^{X_{m+1}})$ is not uniformly Turing reducible to $(L(N_i^{X_{m+1}}), L(N_j^{X_{m+1}}))$ via $M_k^{X_{m+1}}$. This construction ensures that for every i and j, $(L(N_i^X), L(N_j^X))$ is not $\leq_{uT}^{pp}$-complete for DisjNP^X.

We describe the construction of X_{m+1}. We choose some large enough $n = n_m$, and we will add words from Σ_{ij}^{n+1} to the oracle. We need a sufficient number of words in Σ_{ij}^{n+1} for diagonalization. Therefore, n has to be large enough such that

$$r_k(n) p_i(r_k(n))(p_j(r_k(n)) + 1) < 2^n.$$

On the other hand, if $m \geq 2$, then we have to make sure that adding words of length l_{ij}^{n+1} does not influence diagonalizations made in former steps. Therefore, if $m \geq 2$ and $m - 1 = \langle i', j', k' \rangle$, then $n > n_{m-1}$ and n has to be large enough such that l_{ij}^{n+1} is greater than $l_{i'j'}^{n_{m-1}+1}$, $\max(p_{i'}(n_{m-1}), p_{j'}(n_{m-1}))$, and $\max(p_{i'}(r_{k'}(n_{m-1})), p_{j'}(r_{k'}(n_{m-1})))$. Since $n_{m-1} > n_{m-2} > \cdots$, these conditions not only guard against interference with step $m-1$, but guard against interference with all steps $m' < m$.

Suppose there exists an $S \subseteq \Sigma^{n+1}_{ij}$ such that $L(N_i^{X_m \cup S}) \cap L(N_j^{X_m \cup S}) \cap \Sigma^{\leq r_k(n)} \neq \emptyset$. Let $X_{m+1} = X_m \cup S$ and go to the next stage $m+1$.

Otherwise,

$$\text{for all } S \subseteq \Sigma^{n+1}_{ij}, \quad L(N_i^{X_m \cup S}) \cap L(N_j^{X_m \cup S}) \cap \Sigma^{\leq r_k(n)} = \emptyset. \tag{1}$$

In this case, we consider the computation of M_k on 0^n. We determine some $w \in \Sigma^{n+1}_{ij}$ and let $X_{m+1} = X_m \cup \{w\}$. We construct a set $Q \subseteq \overline{L(N_j^{X_{m+1}})}$. Hence $L(N_i^{X_{m+1}}) \cup Q$ is a separator of $(L(N_i^{X_{m+1}}), L(N_j^{X_{m+1}}))$. The sets X_{m+1} and Q satisfy either

$$0^n \in A_{ij}^{X_{m+1}} \quad \text{and} \quad 0^n \notin L(M_k^{L(N_i^{X_{m+1}}) \cup Q}) \tag{2}$$

or

$$0^n \in B_{ij}^{X_{m+1}} \quad \text{and} \quad 0^n \in L(M_k^{L(N_i^{X_{m+1}}) \cup Q}). \tag{3}$$

This shows that $(A_{ij}^{X_{m+1}}, B_{ij}^{X_{m+1}})$ does not $\leq^{pp}_{uT}$-reduce to $(L(N_i^{X_{m+1}}), L(N_j^{X_{m+1}}))$ via M_k.

The difficulty of finding w and Q rises mainly from the following: If we want to preserve the computation of M_k on 0^n, then we have to ensure that all oracle queries are preserved. Since the oracle is a separator of two NP languages, we have to maintain the acceptance behaviors of N_i and N_j with respect to the queries made by $M_k(0^n)$. This results in reserving too many strings. In particular, this may leave no room for the diagonalization in Σ^{n+1}_{ij}. However, by Lemma 3.3, we can do better.

Now we construct the set Q, and at the same time we reserve strings for $\overline{X_{m+1}}$. The latter makes sure that either N_i or N_j rejects on certain queries.

Initially we set $Q = \emptyset$. We run M_k on 0^n using oracle $L(N_i^{X_m}) \cup Q$, until the first string q is queried. We apply Lemma 3.3 with $M = N_i$, $N = N_j$, $Y = X_m$, and $T = \Sigma^{n+1}_{ij}$. By equation (1), the first statement of Lemma 3.3 cannot hold. Hence, there is a set $S' \subseteq \Sigma^{n+1}_{ij}$, $\|S'\| \leq p_i(r_k(n)) \cdot (p_j(r_k(n)) + 1)$, such that either

$$(\forall S,\, S \subseteq \Sigma^{n+1}_{ij},\, S \cap S' = \emptyset)[q \notin L(N_i^{X_m \cup S})] \tag{4}$$

or

$$(\forall S,\, S \subseteq \Sigma^{n+1}_{ij},\, S \cap S' = \emptyset)[q \notin L(N_j^{X_m \cup S})]. \tag{5}$$

We reserve all strings in S' for $\overline{X_{m+1}}$. If equation (4) is true, then we continue running M_k without changing Q. (Hence, answer "no" to query q.) Otherwise, let $Q = Q \cup \{q\}$ and continue running M_k with oracle $X_m \cup Q$. (Hence, answer "yes" to query q.) By the choice of q, Q remains a separator of $(L(N_i^{X_m}), L(N_j^{X_m}))$. We continue running M_k until the next string is queried and then apply Lemma 3.3 again, obtain the set S' that satisfies equation (4) or (5) for the new query, and update Q accordingly. We do this repeatedly until the end of the computation of M_k on 0^n.

The number of strings in Σ^{n+1}_{ij} that are reserved for $\overline{X_{m+1}}$ is at most

$$r_k(n) \cdot p_i(r_k(n)) \cdot (p_j(r_k(n)) + 1) < 2^n.$$

So there exist a string $0^i10^j10x \in \Sigma^{n+1}_{ij}$ and a string $0^i10^j11y \in \Sigma^{n+1}_{ij}$ such that neither string is reserved for $\overline{X_{m+1}}$. If $M_k^{L(N_i^{X_m}) \cup Q}(0^n)$ accepts, then let $w = 0^i10^j11y$.

Otherwise, let $w = 0^i 10^j 10x$. We define $X_{m+1} = X_m \cup \{w\}$. This completes stage m and we can go to the next stage $m+1$.

The following two claims prove the correctness of the construction.

CLAIM 3.9. *After every stage $m = \langle i,j,k\rangle$, either $L(N_i^{X_{m+1}}) \cap L(N_j^{X_{m+1}}) \cap \Sigma^{\leq r_k(n_m)} \neq \emptyset$ or $(A_{ij}^{X_{m+1}}, B_{ij}^{X_{m+1}})$ does not $\leq_{uT}^{pp}$-reduce to $(L(N_i^{X_{m+1}}), L(N_j^{X_{m+1}}))$ via M_k.*

Proof. If $L(N_i^{X_{m+1}}) \cap L(N_j^{X_{m+1}}) \cap \Sigma^{\leq r_k(n_m)} \neq \emptyset$, then we are done. Otherwise, it follows that equation (1) holds. In this case we constructed Q. We know that every string that was added to Q is enforced to be rejected by $N_j^{X_m}$. Since w is not reserved and $X_{m+1} = X_m \cup \{w\}$, Q is also in the complement of $L(N_j^{X_{m+1}})$. Therefore, $L(N_i^{X_{m+1}}) \cup Q$ is a separator of $(L(N_i^{X_{m+1}}), L(N_j^{X_{m+1}}))$.

All queries of $M_k(0^{n_m})$ under oracle $L(N_i^{X_{m+1}}) \cup Q$ are answered the same way as in the construction of Q. The reason is as follows: For any query q, if we reserve strings from $\Sigma_{ij}^{n_m+1}$ for $\overline{X_{m+1}}$ such that N_i always rejects q (equation (4)), then q will not be put into Q. Hence q will get the answer "no" from oracle $L(N_i^{X_{m+1}}) \cup Q$, which is the same as in the construction of Q. If we reserve strings from $\Sigma_{ij}^{n_m+1}$ for $\overline{X_{m+1}}$ such that N_j always rejects q (equation (5)), then q will be put into Q. Hence q gets the answer "yes" under oracle $L(N_i^{X_{m+1}}) \cup Q$, which is the same answer as given in the construction of Q. Therefore, by the choice of w, we obtain the following:

- If $M_k^{L(N_i^{X_{m+1}}) \cup Q}(0^{n_m})$ accepts, then $0^{n_m+1} \in B_{ij}^{L(N_i^{X_{m+1}}) \cup Q}$.
- If $M_k^{L(N_i^{X_{m+1}}) \cup Q}(0^{n_m})$ rejects, then $0^{n_m+1} \in A_{ij}^{L(N_i^{X_{m+1}}) \cup Q}$.

Hence $L(M_k^{L(N_i^{X_{m+1}}) \cup Q})$ does not separate $(A_{ij}^{X_{m+1}}, B_{ij}^{X_{m+1}})$. □

CLAIM 3.10. *For all $(C,D) \in \mathrm{DisjNP}^X$, where $C = L(N_i^X)$ and $D = L(N_j^X)$, it holds that $(A_{ij}^X, B_{ij}^X) \in \mathrm{DisjNP}^X$ and $(A_{ij}^X, B_{ij}^X) \not\leq_{uT}^{pp} (C,D)$.*

Proof. First, we claim that there is no stage $m = \langle i,j,k\rangle$ such that $L(N_i^{X_{m+1}}) \cap L(N_j^{X_{m+1}}) \cap \Sigma^{\leq r_k(n_m)} \neq \emptyset$. Otherwise, since the number n_{m+1} is chosen large enough, all strings that are added to the oracle in later stages will not change the computations of N_i and N_j on inputs of lengths $\leq r_k(n_m)$. Therefore, $L(N_i^X) \cap L(N_j^X) \neq \emptyset$, which contradicts our assumption.

From Claim 3.9 it follows that for every stage $m = \langle i,j,k\rangle$, $(A_{ij}^{X_{m+1}}, B_{ij}^{X_{m+1}})$ does not $\leq_{uT}^{pp}$-reduce to $(L(N_i^{X_{m+1}}), L(N_j^{X_{m+1}}))$ via M_k. Again, since n_{m+1} is chosen large enough, all strings added to the oracle in later stages will not change the following:

1. The membership of 0^{n_m} in $A_{ij}^{X_{m+1}}$ and $B_{ij}^{X_{m+1}}$. Strings of length $l_{ij}^{n_m+1}$ are only added to the oracle at stage m and not in any other stage.
2. The computations of N_i and N_j on inputs of lengths $\leq r_k(n_m)$ (which is the maximal length of strings that can be queried by M_k on 0^{n_m}).

Hence, (A_{ij}^X, B_{ij}^X) does not $\leq_{uT}^{pp}$-reduce to (C,D) via M_k. Since this holds for all k, we obtain $(A_{ij}^X, B_{ij}^X) \not\leq_{uT}^{pp} (C,D)$.

It remains to observe that $(A_{ij}^X, B_{ij}^X) \in \mathrm{DisjNP}^X$: For each $m = \langle i,j,k\rangle$ we added exactly one string from $\Sigma_{ij}^{n_m+1}$ to the oracle. Moreover, for any other $m' = \langle i',j',k'\rangle$ we added only words from $\Sigma_{i'j'}^{n_{m'}+1}$ to the oracle; this does not influence A_{ij}^X and B_{ij}^X. □

This completes the proof of the theorem. □

COROLLARY 3.11. *For the oracle X from Theorem 3.8 it holds that $\mathrm{P}^X \neq \mathrm{UP}^X$.*

Proof. Choose i and j such that N_i^X (resp., N_j^X) accepts X (resp., $\overline{X}$). We show that $A_{ij}^X \in \mathrm{UP}^X - \mathrm{P}^X$.

Note that $L(N_i^X) \cap L(N_j^X) = \emptyset$. By the construction in Theorem 3.11, for every length n, we add at most one string of the form $0^i 10^j 10x$, $|x| = n$, to the oracle. So $A_{ij}^X \in \mathrm{UP}^X$.

Assume $A_{ij}^X = L(M_k^X)$ for some deterministic polynomial-time oracle Turing machine M_k. Note that X is the only separator of $(L(N_i^X), L(N_j^X))$. Therefore, it follows that $(A_{ij}^X, B_{ij}^X) \leq_{uT}^{pp} (L(N_i^X), L(N_j^X))$ via M_k. This contradicts Claim 3.10. □

4. Function classes and disjoint pairs. We show that there exists a Turing-complete disjoint NP-pair if and only if NPSV contains a Turing-complete partial function. We know already that there is a connection between disjoint NP-pairs and NPSV. Namely, Selman [Sel94] proved that Conjecture 2.4 holds if and only if NPSV does not contain an NP-hard partial function, and Köbler and Messner [KM00] proved that there exists a many-one-complete disjoint NP-pair if and only if NPSV contains a many-one-complete partial function. Recall [Sel94] that NPSV is the set of all partial, single-valued functions computed by nondeterministic polynomial-time-bounded transducers.

If g is a single-valued total function, then we define $M[g]$, the single-valued partial function computed by M with oracle g, as follows: $x \in \mathrm{dom}(M[g])$ if and only if M reaches an accepting state on input x. In this case, $M[g](x)$ is the final value of M's output tape. In the case that g is a total function and $f = M[g]$, we write $f \leq_T^p g$.

The literature contains two different definitions of reductions between partial functions, because one must decide what to do in case a query is made to the oracle function when the query is not in the domain of the oracle function. Fenner et al. [FHOS97] determined that in this case the value returned should be a special symbol, $\perp$. Selman [Sel94] permits the value returned in this case to be arbitrary, which is the standard paradigm for promise problems. Here we use the promise problem definition of Selman [Sel94]. Recall that for multivalued partial functions f and g, g is *an extension* of f if $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$, and for all $x \in \mathrm{dom}(f)$ and for every y, if $g(x) \mapsto y$, then $f(x) \mapsto y$.

DEFINITION 4.1. *For polynomial-length-bounded, partial multivalued functions f and g, f is* Turing reducible to g *(as a promise problem, so we write $f \leq_T^{pp} g$) in polynomial time if for some deterministic polynomial-time-bounded oracle transducer M, for every single-valued total extension g' of g, $M[g']$ is an extension of f.*

Here, if the query q belongs to the domain of g, then the oracle returns a value of $g(q)$. We will use the result [Sel94] that $f \leq_T^{pp} g$ if and only if for every single-valued total extension g' of g, there is a single-valued total extension f' of f such that $f' \leq_T^p g'$.

A single-valued partial function g is $\leq_T^{pp}$*-complete* for NPSV if g belongs to NPSV and, for all $f \in$ NPSV, $f \leq_T^{pp} g$.

THEOREM 4.2. NPSV *contains a $\leq_T^{pp}$-complete partial function* $\Leftrightarrow$ DisjNP *contains a $\leq_T^{pp}$-complete pair.*

Proof. For any $f \in$ NPSV, define the following sets:

$$R_f = \{\langle x, y\rangle \mid x \in \mathrm{dom}(f),\ y \leq f(x)\} \tag{6}$$

and

$$S_f = \{\langle x, y\rangle \mid x \in \mathrm{dom}(f),\ y > f(x)\}. \tag{7}$$

Note that (R_f, S_f) is a disjoint NP-pair.

CLAIM 4.3. *For every separator A of (R_f, S_f), there is a single-valued total extension f' of f such that $f' \leq_T^p A$.*

Consider the following oracle transducer T that computes f' with oracle A. On input x, if $x \in \mathrm{dom}(f)$, then T determines the value of $f(x)$, using a binary search algorithm, by making repeated queries to A. Note that for $x \in \mathrm{dom}(f)$ and for any y, if $y \leq f(x)$, then $\langle x, y\rangle \in R_f$, and if $y > f(x)$, then $\langle x, y\rangle \in S_f$. Clearly, T computes some single-valued total extension of f. This proves the claim.

Let f be a $\leq_T^{pp}$-complete function for NPSV and assume that A separates R_f and S_f. By Claim 4.3, there is a single-valued total extension f' of f such that $f' \leq_T^p A$.

Let $(U, V) \in$ DisjNP. We want to show that $(U, V) \leq_T^{pp} (R_f, S_f)$. Define

$$g(x) = \begin{cases} 0 & \text{if } x \in U, \\ 1 & \text{if } x \in V, \\ \uparrow & \text{otherwise.} \end{cases}$$

Then $g \in$ NPSV, so $g \leq_T^{pp} f$. Therefore, there is a single-valued total extension g' of g such that $g' \leq_T^p f'$.

Define $L = \{x \mid g'(x) = 0\}$. It is easy to see that $L \leq_T^p g'$. Also note that $U \subseteq L$ and $V \subseteq \overline{L}$, and, therefore, L separates U and V. Then the following sequence of reductions shows that $L \leq_T^p A$:

$$L \leq_T^p g' \leq_T^p f' \leq_T^p A.$$

Thus, for every separator A of (R_f, S_f), there is a separator L of (U, V) such that $L \leq_T^p A$. Therefore, (R_f, S_f) is $\leq_T^{pp}$-complete for DisjNP.

For the other direction, assume that (U, V) is $\leq_T^{pp}$-complete for DisjNP. Define the following function:

$$f(x) = \begin{cases} 0 & \text{if } x \in U, \\ 1 & \text{if } x \in V, \\ \uparrow & \text{otherwise.} \end{cases}$$

Clearly, $f \in$ NPSV.

Let f' be a single-valued total extension of f, and let $L = \{x \mid f'(x) = 0\}$. Clearly, $L \leq_T^p f'$. Also, since $U \subseteq L$ and $V \subseteq \overline{L}$, L is a separator of (U, V).

We want to show that for any $g \in$ NPSV, $g \leq_T^{pp} f$. Consider the disjoint NP-pair (R_g, S_g) for the function g as defined in equations (6) and (7). There is a separator A of (R_g, S_g) such that $A \leq_T^p L$, since L is a separator of the $\leq_T^{pp}$-complete disjoint NP-pair (U, V). As noted in Claim 4.3, there is a single-valued total extension g' of g such that $g' \leq_T^p A$. Therefore, the following sequence of reductions shows that $g \leq_T^{pp} f$:

$$g' \leq_T^p A \leq_T^p L \leq_T^p f'.$$

Hence, f is complete for NPSV. □

COROLLARY 4.4.

1. *Let $f \in$ NPSV be $\leq_T^{pp}$-complete for* NPSV. *Then (R_f, S_f) is $\leq_T^{pp}$-complete for* DisjNP.

2. *If* (U, V) *is* $\leq_T^{pp}$*-complete for* DisjNP*, then* $f_{U,V}$ *is complete for* NPSV*, where*

$$f_{U,V}(x) = \begin{cases} 0 & \text{if } x \in U, \\ 1 & \text{if } x \in V, \\ \uparrow & \text{otherwise.} \end{cases}$$

3. *Relative to the oracle in Theorem* 3.8*,* NPSV *does not have a* $\leq_T^{pp}$*-complete partial function.*

5. Nonsymmetric pairs and separation of reducibilities. Pudlák [Pud03] defined a disjoint pair (A, B) to be *symmetric* if $(B, A) \leq_m^{pp} (A, B)$. Otherwise, (A, B) is *nonsymmetric*. For example, the canonical disjoint NP-pair for the propositional proof system Resolution is symmetric [Pud03] (see section 6.3 for the definition of canonical pairs). In this section we give complexity-theoretic evidence of the existence of nonsymmetric disjoint NP-pairs. As a consequence, we obtain new ways to demonstrate existence of P-inseparable sets and show that $\leq_m^{pp}$ and $\leq_T^{pp}$ reducibilities differ for disjoint NP-pairs.

A set L is *P-printable* if there is $k \geq 1$ such that all elements of L up to length n can be printed by a deterministic Turing machine in time $n^k + k$ [HY84, HIS85]. Every P-printable set is sparse and belongs to P. An infinite set A is *P-printable-immune* if no infinite subset of A is P-printable.

A set L is *p-selective* if there is a polynomial-time-bounded function f such that for every $x, y \in \Sigma^*$, $f(x, y) \in \{x, y\}$, and $\{x, y\} \cap L \neq \emptyset \Rightarrow f(x, y) \in L$ [Sel79].

A partial function $f \in$ PF is *almost-always one-way* [FPS01] if no polynomial-time Turing machine inverts f correctly on more than a finite subset of range(f).

Proposition 5.1.

1. (A, B) *is symmetric if and only if* (B, A) *is symmetric.*
2. *If* (A, B) *is P-separable, then* (A, B) *is symmetric.*

Proof. The proof of the first assertion is trivial. For the proof of the second assertion, let (A, B) be a P-separable disjoint NP-pair. Fix $a \in A$ and $b \in B$, and let the separator be $S \in$ P. Consider the following polynomial-time computable function f. On input x, if $x \in S$, then f outputs b; otherwise, f outputs a. Therefore, $x \in A$ implies $x \in S$, which implies $f(x) = b \in B$, and $x \in B$ implies $x \notin S$, which implies $f(x) = a \in A$. Therefore, $(A, B) \leq_m^{pp} (B, A)$, i.e., (A, B) is symmetric. □

We will show the existence of a nonsymmetric disjoint NP-pair under certain hypotheses, due to the following proposition, that will separate $\leq_m^{pp}$ and $\leq_T^{pp}$ reducibilities.

Proposition 5.2.

1. *If* (A, B) *is a nonsymmetric disjoint* NP*-pair, then* $(B, A) \not\leq_m^{pp} (A, B)$.
2. *For any disjoint* NP*-pair* (A, B), $(B, A) \leq_T^{pp} (A, B)$.

Proof. The first assertion follows from the definition of symmetric pairs. For the second assertion, observe that for any S separating A and B, $\overline{S}$ separates B and A, while for any set S, $\overline{S} \leq_T^p S$. □

We will use the following proposition in a crucial way to provide some evidence for the existence of nonsymmetric disjoint NP-pairs. In other words, we will seek to obtain a disjoint NP-pair (A, B) such that either A or B is p-selective, but (A, B) is not P-separable.

Proposition 5.3. *For any disjoint* NP*-pair* (A, B)*, if either* A *or* B *is p-selective, then* (A, B) *is symmetric if and only if* (A, B) *is P-separable.*

Proof. We know from Proposition 5.1 that if (A, B) is P-separable, then it is symmetric. Now assume that (A, B) is symmetric via some function f and assume

(without loss of generality) that A is p-selective and the p-selector function is g. The following algorithm M separates A and B. On input x, M runs g on the strings $(x, f(x))$, and accepts x if and only if g outputs x. If $x \in A$, then $f(x) \in B$, and therefore g has to output x. On the other hand, if $x \in B$, then $f(x) \in A$. So g will output $f(x)$ and M will reject x. Therefore, $A \subseteq L(M) \subseteq \overline{B}$. □

Now we give evidence for the existence of nonsymmetric disjoint NP-pairs.

THEOREM 5.4. *If* $\mathrm{E} \neq \mathrm{NE} \cap \mathrm{coNE}$*, then there is a set* $A \in \mathrm{NP} \cap \mathrm{coNP}$ *such that* $(A, \overline{A})$ *is not symmetric.*

Proof. If $\mathrm{E} \neq \mathrm{NE} \cap \mathrm{coNE}$, then there is a tally set $T \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$. From Selman [Sel79, Theorem 5], the existence of such a tally set implies that there is a p-selective set $A \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$. Clearly, $(A, \overline{A})$ is not P-separable. Hence, by Proposition 5.3, $(A, \overline{A})$ is nonsymmetric. □

As a corollary, if $\mathrm{E} \neq \mathrm{NE} \cap \mathrm{coNE}$, then there is a set $A \in \mathrm{NP} \cap \mathrm{coNP}$ such that $(A, \overline{A}) \not\leq_m^{pp} (\overline{A}, A)$, yet clearly $(A, \overline{A}) \leq_T^{pp} (\overline{A}, A)$.

We will show that the hypotheses in Theorem 5.5 imply the existence of a nonsymmetric disjoint NP-pair. Note that the hypotheses in this theorem are similar to those studied by Fortnow, Pavan, and Selman [FPS01] and Pavan and Selman [PS02]. However, our hypotheses are stronger than the former and weaker than the latter.

THEOREM 5.5. *The following are equivalent.*

1. *There is a* UP*-machine* N *that accepts* 0^* *and, for every polynomial-time machine* M*,* $\{n \mid M$ *on input* 0^n *outputs the accepting computation of* N *on input* $0^n\}$ *is a finite set.*
2. *There is a set* S *in* UP *accepted by a* UP*-machine* N *such that* S *has exactly one string of every length and, for every polynomial-time machine* M*, the following set is finite:* $\{n \mid M$ *on input* 0^n *outputs the accepting computation of* N *on input* $x_n\}$*, where* x_n *denotes the word of length* n *that belongs to* S*.*
3. *There is an honest one-to-one, almost-always one-way function* f *such that* $\mathrm{range}(f) = 0^*$*.*
4. *There is a language* $L \in \mathrm{P}$ *that has exactly one string of every length and* L *is P-printable-immune.*
5. *There is a language* $L \in \mathrm{UP}$ *that has exactly one string of every length and* L *is P-printable-immune.*

Proof. We show the following cycles: $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 1$ and $1 \Rightarrow 4 \Rightarrow 5 \Rightarrow 1$.

Trivially, item 1 implies item 2. To prove that item 2 implies item 3, let N be a UP-machine that satisfies the conditions of item 2 and let $S = L(N)$. For any y that encodes an accepting computation of N on some string x, define $f(y) = 0^{|x|}$. Since y also encodes x, f is polynomial-time computable. Since N runs in polynomial time, f is honest. On the other hand, if any polynomial-time computable machine can invert f on 0^n for infinitely many n, then that machine actually outputs infinitely many accepting computations of N.

We show that item 3 implies item 1. Given f as in item 3, we know that since f is honest, $\exists k > 0$ such that $|x| \leq |f(x)|^k$. We describe a UP-machine N that accepts 0^*. On input 0^n, N guesses x, $|x| \leq n^k$, and accepts 0^n if and only if $f(x) = 0^n$. Since f is one-to-one, N has exactly one accepting path for every input of the form 0^n, and since $\mathrm{range}(f) = 0^*$, $L(N) = 0^*$. If there is a polynomial-time machine M that outputs infinitely many accepting computations of N, then M also inverts f on infinitely many strings.

To prove that item 1 implies item 4, let N be the UP-machine in item 1. We can assume without loss of generality that for all but finitely many n, on input 0^n, N has

exactly one accepting computation of length n^k for some $k > 0$. Let us define the following language:

$$L' = \{x10^n10^l \mid n \geq 0,\ x \text{ is an accepting path of } N(0^n), \text{ and } 0 \leq l \leq (n+1)^k - n^k\}.$$

It is easy to see that L' is in P, and for all but finitely many n, L has exactly one string of length n. Therefore, there exists a finite variation $L \in \mathrm{P}$ such that L has exactly one string of every length. If L has an infinite P-printable subset, then so has L'. Let M' be a polynomial-time transducer that prints an infinite subset of L'. It follows that M' outputs infinitely many accepting computations of N.

Item 4 trivially implies item 5. We show that item 5 implies item 1. Let L be such a language in UP via a UP-machine N. Define a UP-machine N' to accept 0^* as follows. On input 0^n, N' guesses a string x of length n and a computation path w of N on x. N' accepts 0^n if and only if w is an accepting computation. If a polynomial-time machine can output infinitely many accepting computations of N', then essentially the same machine also outputs infinitely many strings in L, and hence L cannot be P-printable-immune. □

THEOREM 5.6. *Each of the hypotheses stated in Theorem* 5.5 *implies the existence of nonsymmetric disjoint* NP*-pairs.*

Proof. Let us define the following function:

$$dt(i) = \begin{cases} 1 & \text{if } i = 0, \\ 2^{2^{dt(i-1)}} & \text{otherwise.} \end{cases}$$

Let M be the UP-machine accepting 0^*, as in the first hypothesis in Theorem 5.5. Let a_n be the accepting computation of M on 0^n. We can assume that $|a_n| = p(n)$, where $p(\cdot)$ is some fixed polynomial. We define the following sets:

$$L_M = \{\langle 0^n, w\rangle \mid w \leq a_n,\ n = dt(i) \text{ for some } i > 0\}$$

and

$$R_M = \{\langle 0^n, w\rangle \mid w > a_n,\ n = dt(i) \text{ for some } i > 0\}.$$

Note that (L_M, R_M) is a disjoint NP-pair. We claim that L_M is p-selective. The description of a selector f for L_M follows. Assume that $\langle 0^k, w_1\rangle$ and $\langle 0^l, w_2\rangle$ are input to f. If $k = l$, then f outputs the lexicographically smaller one of w_1 and w_2. Otherwise, assume that $k < l$, and without loss of generality, both k and l are in range(dt). In that case, $l \geq 2^{2^k} > 2^{|a_k|}$, and therefore f can compute a_k, the accepting computation of M on 0^k, by checking all possible strings of length $|a_k|$. Therefore, in $O(l)$ time, f outputs $\langle 0^k, w_1\rangle$ if $w_1 \leq a_k$, and outputs $\langle 0^l, w_2\rangle$ otherwise. Similarly, we can show that R_M is p-selective.

We claim that (L_M, R_M) is a nonsymmetric disjoint NP-pair. Assume on the contrary that this pair is symmetric. Therefore, by Proposition 5.3 (L_M, R_M) is P-separable; i.e., there is $S \in \mathrm{P}$ that is a separator for (L_M, R_M). Using a standard binary search technique, a polynomial-time machine can compute the accepting computation of M on any 0^n, where $n = dt(i)$ for some $i > 0$. Since the length of the accepting computation of M on 0^n is $p(n)$, this binary search algorithm takes time $O(p(n))$ which is polynomial in n. This contradicts our hypothesis, since we assumed that no polynomial-time machine can compute infinitely many accepting computations of M. Therefore, (L_M, R_M) is a nonsymmetric disjoint NP-pair. □

If the hypotheses stated in Theorem 5.5 hold, then there exists a disjoint NP-pair (A, B) so that $(A, B) \not\leq_m^{pp} (B, A)$ while $(A, B) \leq_T^{pp} (B, A)$.

Grollmann and Selman [GS88] proved that the existence of P-inseparable disjoint NP-pairs implies the existence of P-inseparable pairs where both sets of the pair are NP-complete. The following results are in the same spirit. We note that natural candidates for nonsymmetric (or $\leq_m^{pp}$-complete) disjoint NP-pairs arise either from cryptography or from proof systems [Pud03]. However, the following theorems show that the existence of such pairs will imply that nonsymmetric (or $\leq_m^{pp}$-complete) disjoint NP-pairs exist where both sets of the pair are $\leq_m^p$-complete for NP.

THEOREM 5.7. *There exists a nonsymmetric disjoint* NP*-pair* (A, B) *if and only if there exists a nonsymmetric disjoint* NP*-pair* (C, D) *where both* C *and* D *are* $\leq_m^p$*-complete for* NP.

Proof. The *if* part is trivial. We prove the *only if* part. Let $\{NM_i\}_{i\geq 1}$ be a standard enumeration of polynomial-time-bounded nondeterministic Turing machines with associated polynomial-time bounds $\{p_i\}_{i\geq 1}$. It is known that the following set is NP-complete [BGS75]:

$$K = \{\langle i, x, 0^n\rangle \mid NM_i \text{ accepts } x \text{ within } n \text{ steps}\}.$$

Let (A, B) be a nonsymmetric disjoint NP-pair. There exists $i \geq 1$ such that $A = L(NM_i)$, and $A \leq_m^p K$ via $f(x) = \langle i, x, 0^{p_i(|x|)}\rangle$. Note that f is honest and one-to-one.

Our first goal is to show that $(K, f(B))$ is nonsymmetric. Since f is a reduction from A to K and $A \cap B = \emptyset$, $f(A) \subseteq K$ and $f(B) \subseteq \overline{K}$, and so $f(B)$ and K are disjoint sets. Observe that $f(B)$ is in NP because on any input y, we can guess x, and verify that $x \in B$ and $f(x) = y$. Therefore, $(K, f(B))$ is a disjoint NP-pair, and K is $\leq_m^p$-complete for NP.

In order to prove that this pair is nonsymmetric, assume otherwise. Then $(K, f(B)) \leq_m^{pp} (f(B), K)$ and, therefore, $\exists g \in$ PF such that $g(K) \subseteq f(B)$ and $g(f(B)) \subseteq K$. Consider the following polynomial-time computable function h. On input x, h first computes $y = g(f(x))$. If $y = \langle i, x', 0^{p_i(|x'|)}\rangle$ for some x', then h outputs x'; otherwise, it returns a fixed string $a \in A$. We claim that $h(A) \subseteq B$ and $h(B) \subseteq A$, thereby making (A, B) symmetric. For any $x \in A$, we know that $f(x) \in K$. Hence $g(f(x)) \in f(B)$, since $g(K) \subseteq f(B)$. So $g(f(x)) = \langle i, x', 0^{p_i(|x'|)}\rangle$ for some $x' \in B$, and so $h(x) = x' \in B$. For any $x \in B$, $y = g(f(x)) \in K$, since $g(f(B)) \subseteq K$. If $y = \langle i, x', 0^{p_i(|x'|)}\rangle$ for some x', then x' must be in A; else h will return $a \in A$, and so, in either case, $x \in B$ will imply that $h(x) \in A$. Therefore, $h(A) \subseteq B$ and $h(B) \subseteq A$. Thus $(A, B) \leq_m^{pp} (B, A)$, contradicting the fact that (A, B) is nonsymmetric. Hence $(K, f(B))$ is a nonsymmetric disjoint NP-pair.

To complete the proof of the theorem, apply the construction once again, this time with an honest reduction f' from $f(B)$ to K. Namely, $f'(f(B)) \subseteq K$ and $f'(K) \subseteq \overline{K}$. Similar to the above argument, it can be shown that $f'(K)$ and K are disjoint. Also, since f' is one-to-one, we claim that $f'(K)$ is $\leq_m^p$-complete for NP. Clearly, $x \in K$ implies $f'(x) \in f'(K)$. On the other hand, for some $x \notin K$, $f'(x)$ cannot be in $f'(K)$; otherwise, $f'(x) = f'(y)$ for some $y' \in K$, contradicting the fact that f' is one-to-one. Then K and $f'(K)$ are disjoint NP-complete sets, and the argument already given shows that $(f'(K), K)$ is nonsymmetric. □

THEOREM 5.8. *There exists a* $\leq_m^{pp}$*-complete disjoint* NP*-pair* (A, B) *if and only if there exists a* $\leq_m^{pp}$*-complete disjoint* NP*-pair* (C, D)*, where both* C *and* D *are* $\leq_m^p$*-complete sets for* NP.

Proof. The proof is similar to that of Theorem 5.7. Consider the one-to-one

function f defined by $f(x) = \langle i, x, 0^{p_i(|x|)} \rangle$ that many-one reduces A to the canonical NP-complete set K.

Obviously $(A, B) \leq_m^{pp} (K, f(B))$ via f, since $f(A) \subseteq K$, and $K \cap f(B) = \emptyset$, as shown in the proof of Theorem 5.7. Similar to that theorem, we apply the one-to-one function f' that many-one reduces $f(B)$ to K to obtain another disjoint pair $(f'(K), K)$ where $(K, f(B)) \leq_m^{pp} (f'(K), K)$ via f'. So $(A, B) \leq_m^{pp} (K, f(B)) \leq_m^{pp} (f'(K), K)$. Therefore $(f'(K), K)$ is also a $\leq_m^{pp}$-complete disjoint NP-pair, and both $f'(K)$ and K are $\leq_m^p$-complete sets for NP. □

6. Optimal proof systems relative to an oracle. The question of whether optimal propositional proof systems exist has been studied in detail. Pudlák [Pud86] and Krajíček and Pudlák [KP89] showed that NE = coNE implies the existence of optimal proof systems. Ben-David and Gringauze [BDG98] and Köbler, Messner, and Torán [KMT03] obtained the same conclusion under weaker assumptions. On the other hand, Messner and Torán [MT98] and Köbler, Messner, and Torán [KMT03] proved that existence of optimal proof systems results in the existence of $\leq_m^p$-complete sets for the promise class NP ∩ SPARSE. These results hold relative to all oracles. Therefore, optimal proof systems exist relative to any oracle in which NE = coNE holds. Krajíček and Pudlák [KP89], Ben-David and Gringauze [BDG98], and Buhrman et al. [BFFvM00] constructed oracles relative to which optimal proof systems do not exist. In addition, NP ∩ SPARSE does not have complete sets relative to the latter oracle.

The relationship between the existence of optimal proof systems and disjoint NP-pairs was first established by Razborov [Raz94], who showed that the existence of optimal proof systems implies the existence of many-one-complete disjoint NP-pairs. Köbler, Messner, and Torán [KMT03] proved that this holds even for a stronger form of many-one reductions. They defined *strong many-one reduction* (we denote this by $\leq_{sm}^{pp}$) between disjoint NP-pairs as follows: $(A, B) \leq_{sm}^{pp} (C, D)$ if there is $f \in$ PF such that $f(A) \subseteq C$, $f(B) \subseteq D$, and $f(\overline{A \cup B}) \subseteq \overline{C \cup D}$.[1]

In this section, we construct two oracles, O_1 and O_2. Relative to O_1, NE = coNE, and therefore [Pud86, KP89] optimal proof systems exist, implying the existence of $\leq_m^p$-complete sets for NP ∩ SPARSE [MT98] as well as the existence of $\leq_{sm}^{pp}$-complete disjoint NP-pairs [KMT03]. On the other hand, relative to this oracle, E ≠ NE∩coNE = NE, thus implying, by Theorem 5.4, that nonsymmetric (and therefore P-inseparable) pairs exist. Since nonexistence of $\leq_T^{pp}$-complete disjoint NP-pairs implies Conjecture 2.4, it is natural to ask whether the converse of this implication holds. Relative to O_1, Conjecture 2.4 holds, and so the converse is false.

Ben-David and Gringauze [BDG98] asked whether the converse to Razborov's result holds. Relative to O_2, NP ∩ SPARSE does not have a complete set, and so optimal proof systems do not exist. On the other hand, $\leq_{sm}^{pp}$-complete disjoint NP-pairs exist. This shows that the converse to Razborov's result does not hold (even for the stronger notion of many-one reduction) in a relativized setting. Relative to O_2, the existence of $\leq_{sm}^{pp}$-complete disjoint NP-pairs does not imply the existence of $\leq_m^p$-complete sets in NP∩SPARSE. In addition, relative to O_2, NE ≠ coNE [Pud86, KP89] and nonsymmetric disjoint NP-pairs exist.

Since relative to both O_1 and O_2, Conjecture 2.4 holds, $\leq_{sm}^{pp}$-complete disjoint NP-pairs exist, and nonsymmetric pairs exist, it follows that these are "independent"

[1] A forthcoming paper [GSS04] proves that there exist $\leq_{sm}^{pp}$-complete disjoint NP-pairs if and only if there exist $\leq_m^{pp}$-complete disjoint NP-pairs.

TABLE 1
Comparison of oracle properties.

	O_1	O_2
$\exists \leq_{sm}^{pp}$-complete disjoint NP-pairs	Yes	Yes
$\exists$ nonsymmetric disjoint NP-pairs	Yes	Yes
Conjecture 2.4 holds	Yes	Yes
E $\neq$ NE	Yes	Yes
NE = coNE	Yes	No
$\exists$ optimal propositional proof systems	Yes	No
NP $\cap$ SPARSE has $\leq_m^p$-complete sets	Yes	No

of the assertion that NE = coNE, the existence of optimal proof systems, and existence of $\leq_m^p$-complete sets in NP $\cap$ SPARSE. In Table 1, we summarize the properties of both oracles; "Yes" denotes that a particular property holds, while "No" means that the property does not hold.

6.1. Notation. We fix the following enumerations: $\{NM_i\}_i$ is an effective enumeration of nondeterministic, polynomial-time-bounded oracle Turing machines; $\{NE_i\}_i$ is an effective enumeration of nondeterministic, linear exponential-time-bounded oracle Turing machines; $\{M_i\}_i$ is an effective enumeration of deterministic, polynomial-time-bounded oracle Turing machines; $\{E_i\}_i$ is an effective enumeration of deterministic, linear exponential-time-bounded oracle Turing machines; and $\{T_i\}_i$ is an effective enumeration of deterministic, polynomial-time-bounded oracle Turing transducers. Moreover, NM_i, M_i, and T_i have running time $p_i = n^i$, and NE_i and E_i have running time 2^{in} independent of the choice of the oracle. For any oracle Z, let f_i^Z denote the function that T_i^Z computes.

We use the following model of nondeterministic oracle Turing machines. On some input the machine starts the first phase of its computation, during which it is allowed to make nondeterministic branches. In this phase the machine is not allowed to ask any queries. At the end of the first phase the machine has computed a list of queries $q_1, \ldots, q_n$, a list of guessed answers $g_1, \ldots, g_n$, and a character, which is either + or −. Now the machine asks in parallel all queries and gets the vector of answers $a_1, \ldots, a_n$. The machine accepts if the computed character is + and $(a_1, \ldots, a_n) = (g_1, \ldots, g_n)$; otherwise the machine rejects. An easy observation shows that for every nondeterministic polynomial-time oracle Turing machine M there exists a machine N that works in the described way such that for all oracles X, $L(M^X) = L(N^X)$.[2] The analogous statement holds for nondeterministic, linear exponential-time-bounded oracle Turing machines.

A computation path P of a nondeterministic polynomial-time oracle Turing machine N on an input x contains all nondeterministic choices, all queries, and all guessed answers. A computation path P that has the character + (resp., −) is called a positive (resp., negative) path. The set of queries that are guessed to be answered positively (resp., negatively) is denoted by P^{yes} (resp., P^{no}); the set of all queries is denoted by $P^{\text{all}} \stackrel{df}{=} P^{\text{yes}} \cup P^{\text{no}}$. The length of P (i.e., the number of computation steps) is denoted by $|P|$. Note that this description of paths makes it possible to talk about paths of computations without specifying the oracle; i.e., we can say that N on x has

[2]Note that for this property we need both: the character must be + and g_i must be guessed correctly. If the machine accepts just when the answers are guessed correctly, then we miss the machine that accepts $\emptyset$ for every oracle.

a positive path P such that P^{yes} and P^{no} satisfy certain conditions. However, when talking about accepting and rejecting paths we always have to specify the oracle. (A positive path can be accepting for certain oracles, and it can be rejecting for other oracles.)

For $X, Y \subseteq \Sigma^*$ we write $Y \supseteq_m X$ if $X \subseteq \Sigma^{\leq m}$ and $Y^{\leq m} = X$. We write $Y \subseteq_m X$ if and only if $X \supseteq_m Y$. We need to consider injective, partial functions $\mu : \mathbb{N}^+ \to \mathbb{N} \times \mathbb{N}^+$ that have a finite domain. We do not distinguish between the function and the set of all (n, i, j) such that $\mu(n) = (i, j)$. We denote both by μ. Let μ and μ' be injective, partial functions $\mathbb{N}^+ \to \mathbb{N} \times \mathbb{N}^+$ that have a finite domain. If $\mu \neq \emptyset$, then $\mu_{\max} \stackrel{df}{=} \max(\text{dom}(\mu))$. We write $\mu \preceq \mu'$ if either $\mu = \emptyset$, or $\mu \subseteq \mu'$ and $\mu_{\max} < n$ for all $n \in \text{dom}(\mu' - \mu)$. We write $\mu \prec \mu'$ if $\mu \preceq \mu'$ and $\mu \neq \mu'$.

For $j \geq 1$, SPARSE_j denotes the class of all languages L such that for all $k \geq 0$, $\|L \cap \Sigma^k\| \leq k^j + j$.

6.2. Existence of optimal proof systems. Now we develop the first of these oracles.

THEOREM 6.1. *There exists an oracle relative to which the following holds:*

(i) $\text{E} \neq \text{NE} = \text{coNE}$.

(ii) *Conjecture* 2.4 *holds.*

For a fixed set X, let us define the following set, which is complete for NE^X:

$$C^X \stackrel{df}{=} \{\langle i, x, l\rangle \mid NE_i^X \text{ accepts } x \text{ within } l \text{ steps}\}.$$

We also define the following property:

$$\text{P1:} \qquad \langle i, x, l\rangle \in C^X \Leftrightarrow (\forall y,\ |y| = 2^{2|\langle i,x,l\rangle|})[\langle i, x, l\rangle y \notin X].$$

We call a set $X \subseteq \Sigma^{\leq k}$ *k-valid* if property P1 holds for all strings $\langle i, x, l\rangle$ such that $|\langle i, x, l\rangle| + 2^{2|\langle i,x,l\rangle|} \leq k$. Note that $\emptyset$ is 0-valid and that the condition on the right-hand side of P1 only depends on words in X that have length $2^{2n} + n$ for some natural number n. We define the following sets:

$$A^X \stackrel{df}{=} \{0^n \mid (n \text{ is odd}) \wedge (\exists y,\ |y| = 2^n)[y \in X]\}$$

and

$$B^X \stackrel{df}{=} \{0^{2^n} z \mid (n \text{ is odd}) \wedge |z| = 2^n \wedge (\exists y,\ |y| = 2^n)[zy \in X]\}.$$

Clearly, $A^X \in \text{NE}^X$ and $B^X \in \text{NP}^X$. We require the following for O_1:

1. $C^{O_1} \in \text{coNE}^{O_1}$. (This implies $\text{NE}^{O_1} = \text{coNE}^{O_1}$, because C^{O_1} is complete for NE^{O_1} by a reduction that is computable in linear time.)
2. $A^{O_1} \notin \text{E}^{O_1}$ (which implies $\text{E}^{O_1} \neq \text{NE}^{O_1}$, since $A^{O_1} \in \text{NE}^{O_1}$).
3. For every i, j, and r, B^{O_1} does not $\leq_T^{pp}$-reduce to $(L(NM_i^{O_1}), L(NM_j^{O_1}))$ via M_r. This will ensure that Conjecture 2.4 holds relative to O_1.

Proof of Theorem 6.1. We will begin by stating two lemmas that will be used in this proof.

LEMMA 6.2. *For every i and every k-valid X, there exists an l-valid $Y \supseteq_k X$, where $l > k$, such that for every $Z \supseteq_l Y$, $A^Z \neq L(E_i^Z)$.*

LEMMA 6.3. *For every i, j, r and every k-valid X, there exists an l-valid $Y \supseteq_k X$, where $l > k$, such that for every $Z \supseteq_l Y$, B^Z does not $\leq_T^{pp}$-reduce to $(L(NM_i^Z), L(NM_j^Z))$ via M_r.*

We define the following list $\mathcal{T}$ of requirements. At the beginning of the construction, $\mathcal{T}$ contains $\{i\}_{i \geq 1}$ and $\{(i, j, r)\}_{i,j,r \geq 1}$. These have the following interpretations:

- $i \in \mathcal{T}$: ensure that $A^{O_1} \neq L(E_i^{O_1})$.
- $(i,j,r) \in \mathcal{T}$: ensure that B^{O_1} does not $\leq_T^{pp}$-reduce to $(L(NM_i^{O_1}), L(NM_j^{O_1}))$ via M_r.

The following algorithm is used to construct the oracle O_1.

```
1    O_1 := ∅; k := 0
2    while {true} {
3        Remove the next requirement t from T
4        if t = i then
5           apply Lemma 6.2 with X = O_1 to get Y and l
6        else  // t = (i, j, r)
7           apply Lemma 6.3 with X = O_1 to get Y and l
8        O_1 := Y; k := l
9    }
```

It is clear that the oracle constructed by this algorithm satisfies items (i) and (ii) of Theorem 6.1. It remains to prove Lemmas 6.2 and 6.3.

Proof of Lemma 6.2. Fix an i and let X be any k-valid oracle. Let n be the smallest odd length such that $k \leq 2^n - 1$, $n-1 < 2^{n-1}$, and $2^{in} < 2^{2^n}$. Note first that we can assume that $k = 2^n - 1$. Otherwise, we claim that X can be extended to some (2^n-1)-valid oracle $X' \supseteq_k X$. Assume that X is $(m-1)$-valid for $k < m \leq 2^n - 1$; we will show how X can be extended to an m-valid oracle. This can be iterated to extend X to be (2^n-1)-valid.

Assume $m = 2^{2r} + r$ and consider some $\langle j,x,l\rangle$ of length r. (If m is not of this form, then, by property P1, an $(m-1)$-valid oracle is automatically an m-valid oracle.)

Note that $|x| \leq r$ and $|l| \leq r$. Hence, $NE_j^X(x)$ can ask only queries of length $\leq 2^r < m-1$. The answers to these queries will not change during the later stages of the construction. So the result of $NE_j^X(x)$ is fixed. If $NE_j^X(x)$ rejects within l steps, then choose some y of length 2^{2r} and put $\langle j,x,l\rangle y$ in X. Otherwise, do not put any such string in X. After all strings $\langle j,x,l\rangle$ are treated, we obtain an oracle X that is m-valid. This shows that we can assume X to be (2^n-1)-valid.

Also note that any string $w = \langle j,x,l\rangle y$ cannot have length 2^n. If $|w| = 2^n$, then, since $|y| = 2^{2|\langle j,x,l\rangle|}$, $|\langle j,x,l\rangle| < n/2$. Hence, the highest length possible for $\langle j,x,l\rangle$ is $(n-1)/2$, in which case $|y| = 2^{n-1}$ and $|w| = (n-1)/2 + 2^{n-2} < 2^n$. If $|\langle j,x,l\rangle|$ is even smaller, then y is of smaller length as well, and so is $|w|$. This shows that $|w|$ can never be 2^n for any n. As a consequence, we know that at stage $k+1$ we do not have to put any strings of the form $\langle j,x,l\rangle y$ into X. Therefore, we can use this stage for diagonalization.

Now we want to show that there exists an l-valid Y, $l \geq 2^n$, such that for every $Z \supseteq_l Y$, $A^Z \neq L(E_i^Z)$. Consider the computation of E_i^X on 0^n. Since the running time of E_i is bounded above by 2^{in}, the queries made by $E_i^X(0^n)$ have length at most 2^{in}. Let N be the set of queries of length $\geq 2^n$ (these are answered "no" in this computation). Note that $\|N\| \leq 2^{in} < 2^{2^n}$. We put some $v \in \Sigma^{2^n} - N$ in X if and only if $E_i^X(0^n)$ rejects. By the above discussion, $k = 2^n \neq 2^{2r} + r$ for any r, and so v cannot be of the form $\langle j,x,l\rangle y$. Therefore, X is 2^n-valid.

Claim 6.4. *We can extend X to some 2^{in}-valid $Y \supseteq_{2^n} X$ such that $N \subseteq \overline{Y}$.*

Proof. Fix some $\langle j,x,l\rangle$ such that $2^n < |\langle j,x,l\rangle y| \leq 2^{in}$. First we show that there are at least 2^{2^n} different such y for this $\langle j,x,l\rangle$. We show this by proving that $|y| \geq 2^n$. If $|y| < 2^n$, then, since length of y can only be a power of 2, let us assume that $y = 2^{n-1}$. Then $|\langle j,x,l\rangle| = (n-1)/2$, and therefore $|\langle j,x,l\rangle y| = (n-1)/2 + 2^{n-1} < 2^n$, contradicting that $|\langle j,x,l\rangle y| > 2^n$.

Now, simulate $NE_j^X(x)$ for l steps. If the simulation $NE_j^X(x)$ accepts within l steps, then do not update X. Otherwise, i.e., if the simulation rejects, then choose y' such that $|y'| = 2^{2|\langle j,x,l\rangle|}$ and $\langle j,x,l\rangle y' \notin N$. Put $\langle j,x,l\rangle y'$ in X. Existence of such y' is ensured, since the possible number of these words is 2^{2^n}, whereas $\|N\| \le 2^{in} < 2^{2^n}$.

So, if NE_j^X accepts x within l steps, no extra string is put in X. On the other hand, if $NE_j^X(x)$ does not accept within l steps, then we put an appropriate $\langle j,x,l\rangle y' \notin N$ in X. Once this procedure is completed for all $\langle j,x,l\rangle$, the oracle we obtain is 2^{in}-valid. We call that oracle Y. This proves Claim 6.4. □

The proof of the lemma is completed by noting that $Y \supseteq_{2^n} X$ and $Y \subseteq \overline{N}$. Hence, $0^n \in A^Y \Leftrightarrow 0^n \notin L(E_i^Y)$. Let $l = 2^{in}$ (which is the l Lemma 6.2 refers to). Any $Z \supseteq_{2^{in}} Y$ differs from Y only by strings of lengths $> 2^{in}$. This does not affect the computation of $E_i(0^n)$, and therefore, by our construction, it follows that $0^n \in A^Z \Leftrightarrow 0^n \notin L(E_i^Z)$. This proves Lemma 6.2. □

Proof of Lemma 6.3. Similar to the proof of Lemma 6.2, we can assume that $k = 2^{n+1} - 1$, where n is odd. Let $c \stackrel{df}{=} (2^{n+1})^{r(i+j)}$. We choose n to be large enough so that the following hold:

- $p_r(2^{n+1})p_i(p_r(2^{n+1}))(p_j(p_r(2^{n+1})) + 1) < 2^{2^n}$;
- $2(2^{n+1})^{2r(i+j)} < 2^{2^n}$, i.e., $2c^2 < 2^{2^n}$.

CLAIM 6.5. *There exist $Y' \subseteq \Sigma^{\le c}$, $N' \subseteq \Sigma^{\le c}$ such that $\|Y'\| \le c^2$, $\|N'\| \le c^2$, and for all $X' \subseteq \Sigma^{2^{n+1}}$, if $N' \subseteq \overline{X'}$, then $X \cup Y' \cup X'$ is c-valid.*

We will prove this claim later.

Choose some z such that $|z| = 2^n$ and for all y, $|y| = 2^n$, $zy \notin Y'$, and $zy \notin N'$. (Such a z exists because both $\|Y'\|, \|N'\| \le c^2$, and $2c^2 < 2^{2^n}$.) We can assume that

$$(\forall X' \subseteq z\Sigma^{2^n})[L(NM_i^{X\cup Y'\cup X'}) \cap L(NM_j^{X\cup Y'\cup X'}) \cap \Sigma^{\le p_r(2^{n+1})} = \emptyset]. \tag{8}$$

Otherwise $Y = X \cup Y' \cup X'$ satisfies the requirement of Lemma 6.3.

We will consider the computation of M_r on $0^{2^n}z$ and construct sets Q and X' such that $L(NM_i^{X\cup Y'\cup X'}) \cup Q$ is a separator of $L(NM_i^{X\cup Y'\cup X'})$ and $L(NM_j^{X\cup Y'\cup X'})$, and either

$$0^{2^n}z \in B^{X\cup Y'\cup X'} \quad \text{and} \quad 0^{2^n}z \notin L(M_r^{L(NM_i^{X\cup Y'\cup X'})\cup Q})$$

or

$$0^{2^n}z \notin B^{X\cup Y'\cup X'} \quad \text{and} \quad 0^{2^n}z \in L(M_r^{L(NM_i^{X\cup Y'\cup X'})\cup Q}).$$

This will imply that $B^{X\cup Y'\cup X'}$ does not $\le_T^{pp}$-reduce to $(L(NM_i^{X\cup Y'\cup X'}), L(NM_j^{X\cup Y'\cup X'}))$ via M_r. The details follow.

Initially we set $Q = \emptyset$. We run M_r on $0^{2^n}z$ using oracle $L(NM_i^{X\cup Y'}) \cup Q$. Note that this oracle is a separator of $(L(NM_i^{X\cup Y'}), L(NM_j^{X\cup Y'}))$. The simulation of M_r on $0^{2^n}z$ is continued until it makes some query q. At this point, we apply Lemma 3.3 with $M = NM_i$, $N = NM_j$, $Y = X \cup Y'$, and $T = z\Sigma^{2^n}$. Note that on input $0^{2^n}z$, M_r can make queries up to length $p_r(2^{n+1})$, and we have $\|T\| = 2^{2^n} > p_i(p_r(2^{n+1}))(p_j(p_r(2^{n+1})) + 1)$. By Lemma 3.3 and equation (8), there is a set $S' \subseteq z\Sigma^{2^n}$ such that either

$$(\forall S \subseteq z\Sigma^{2^n},\ S \cap S' = \emptyset)[q \notin L(NM_i^{X\cup Y'\cup S})] \tag{9}$$

or

$$(\forall S \subseteq z\Sigma^{2^n},\ S \cap S' = \emptyset)[q \notin L(NM_j^{X\cup Y'\cup S})]. \tag{10}$$

We know that $\|S'\| \leq p_i(p_r(2^{n+1}))(p_j(p_r(2^{n+1})) + 1)$. We reserve all strings in S' for $\overline{X'}$. If equation (9) is true, then we continue simulating M_r without modifying the oracle (hence, answer "no" to query q). Otherwise, if equation (9) does not hold, we update $Q = Q \cup \{q\}$ (hence, answer "yes" to query q and add q to the oracle) and continue the simulation of M_r on $0^{2^n}z$. We continue running M_r until the next query, and then we apply Lemma 3.3 again, obtain the set S' that satisfies above equation (9) or equation (10) for the new query and update Q accordingly. We keep doing this until the end of the computation of M_r on $0^{2^n}z$. The number of strings in $z\Sigma^{2^n}$ we reserved for $\overline{X'}$ during the above process is at most $p_r(2^{n+1})p_i(p_r(2^{n+1}))(p_j(p_r(2^{n+1})) + 1) < 2^{2^n}$ since the running time of M_r on $0^{2^n}z$ is bounded by $p_r(2^{n+1})$.

Since the number of strings reserved for $\overline{X'}$ in the above process is strictly less than the number of strings of length 2^n, there exists a string zy in $z\Sigma^{2^n}$ that is not reserved for $\overline{X'}$. If M_r using oracle $L(NM_i^{X\cup Y'}) \cup Q$ accepts $0^{2^n}z$, we define $X' = \emptyset$. In this case, $0^{2^n}z \notin B^{X\cup Y'\cup X'}$. Otherwise, define $X' = \{zy\}$, in which case $0^{2^n}z \in B^{X\cup Y'\cup X'}$. Also observe that q is put in Q only when $q \notin L(NM_j^{X\cup Y'\cup X'})$. Therefore, $L(NM_i^{X\cup Y'\cup X'}) \cup Q$ remains a separator of $L(NM_i^{X\cup Y'\cup X'})$ and $L(NM_j^{X\cup Y'\cup X'})$.

Let $Y \stackrel{df}{=} X \cup Y' \cup X'$. It is clear from the discussion above that B^Y does not $\leq_T^{pp}$-reduce to $L(NM_i^Y, NM_j^Y)$ via M_r. Since $X' \subseteq \overline{N'}$, Y is $c = (2^{n+1})^{r(i+j)}$-valid. Furthermore, any string q that can be queried by M_r on $0^{2^n}z$ is of length $\leq (2^{n+1})^r$. Therefore, the strings that are queried by NM_i and NM_j on input q are of lengths at most $(2^{n+1})^{r(i+j)} = c$. This implies that for all $Z \supseteq_c Y$, B^Z does not $\leq_T^{pp}$-reduce to $(L(NM_i^Z), L(NM_j^Z))$ via M_r, since any string of length more than c will not affect the outcome of the computation. It remains to prove Claim 6.5.

Proof of Claim 6.5. We use the following algorithm to construct Y' and N'. Recall that $c = (2^{n+1})^{r(i+j)}$.

```
1.   Y' = ∅, N' = ∅
2.   Treated = ∅
3.   L = {⟨i,x,l⟩ | 2^{n+1} < |⟨i,x,l⟩y| ≤ c where |y| = 2^{2|⟨i,x,l⟩|}}
4.   while L ≠ ∅ {
5.      Remove the smallest ⟨i,x,l⟩ from L
6.      Treated = Treated ∪ {⟨i,x,l⟩}
7.      if (∃X' ⊆ Σ^{2^{n+1}} such that X' ⊆ N̄' and
                    NE_i^{X∪Y'∪X'}(x) accepts within l steps)
8.         Choose an accepting path P
9.         Y' = Y' ∪ P^yes and N' = N' ∪ P^no
        else
10.        Choose some y ∈ Σ^{2|⟨i,x,l⟩|} such that ⟨i,x,l⟩y ∉ N'
11.        Y' = Y' ∪ {⟨i,x,l⟩y}
12.  } //end while.
```

We claim that after each iteration of the *while* loop, the following invariance holds: For every $X' \subseteq \overline{N'} \cap \Sigma^{2^{n+1}}$, property P1 holds for each $\langle i, x, l\rangle$ in `Treated` with oracle $X \cup Y' \cup X'$. Initially, when `Treated` is empty, this holds trivially.

Let us assume that $\langle i, x, l\rangle$ is put in `Treated` during iteration $m \geq 1$ of the while loop. It is straightforward to see that after this iteration, the statements in the loop ensure that the invariance holds for $\langle i, x, l\rangle$, since $\langle i, x, l\rangle y$ is put into the oracle if and only if NE_i does not accept x within l steps. We have to show that the invariance also holds for every such triple that had been put into `Treated` in some iteration $m' < m$. Let $\langle j, u, t\rangle$ be such a triple. It suffices to show that for t steps, $NE_j(u)$ behaves the

same way after the mth iteration as it does after the m'th iteration. Assume that during the m'th iteration NE_j accepted u in t steps. All the queries that are made on that accepting path are already in Y' or N' accordingly. Therefore, that path remains accepting even during the mth iteration.

On the other hand, let us assume that for every X', NE_j rejected u in t steps during the m'th iteration. We will show that it will still reject u after the mth iteration. To see this, let us assume that a previously rejecting path has become an accepting path after the mth iteration. A query that was answered "yes" at that point cannot be answered "no" now, since Y' now contains strictly more strings. So assume that the queries $q_1, \ldots, q_d$ were answered "no" during the m'th iteration with $X \cup Y' \cup X'$ as the oracle and are now answered "yes." All strings that are added to Y' after iteration m' are either of lengths $\geq |\langle j, u, t\rangle y| > t$ or are from some $X' \subseteq \Sigma^{2^{n+1}}$. Hence $q_1, \ldots, q_d$ must be of length 2^{n+1}. Note that at least one of these queries must have been in N' during the m'th iteration; otherwise NE_j would accept u at that point with oracle $X \cup Y' \cup (X' \cup \{q_1, \ldots, q_d\})$. But any string that was in N' during an earlier iteration is not put in X' or Y' in later iterations. Therefore, our assumption is false, and NE_j will reject u during the mth iteration as well. This proves the invariance.

What remains to show are the bounds on the sizes of Y' and N' and the maximum length of strings in Y' and N'. For the size of Y' and N', note that if $|\langle i, x, l\rangle y| \leq c$, then, since $|y| = 2^{2|\langle i,x,l\rangle|}$, $|\langle i, x, l\rangle| \leq (\log c)/2$, and therefore $\|\mathcal{L}\| \leq 2^{(\log c)/2+1} < c$. On the other hand, during every iteration, at most l strings are added to Y' and N', and $|l| < |\langle i, x, l\rangle| \leq (\log c)/2$, and therefore $l < c$ as well. Since both Y' and N' are initially empty, they are at most c^2 in size. The maximum length of strings in Y' and N' is c since the longest string that is added to Y' or N' is $\max_{\langle i,x,l\rangle \in \mathcal{L}} |\langle i, x, l\rangle y| \leq c$.

This completes the proof of Claim 6.5. □

This finishes the proof of Lemma 6.3. □

This proves Theorem 6.1. □

COROLLARY 6.6. *The oracle O_1 of Theorem* 6.1 *has the following additional properties:*

(i) $\mathrm{UP}^{O_1} \neq \mathrm{NP}^{O_1} \neq \mathrm{coNP}^{O_1}$ *and* $\mathrm{NPMV}^{O_1} \not\subseteq_c \mathrm{NPSV}^{O_1}$.

(ii) *Relative to O_1, optimal propositional proof systems exist.*

(iii) *There exists a $\leq_{sm}^{pp,O_1}$-complete disjoint NP^{O_1}-pair (A, B) that is P^{O_1}-inseparable but symmetric.*

6.3. Nonexistence of optimal proof systems. In this section we construct an oracle relative to which there exist $\leq_{sm}^{pp}$-complete disjoint NP-pairs. For any oracle X, $(A, B) \leq_{sm}^{pp,X} (C, D)$ if there is a function $f \in \mathrm{PF}^X$ such that $f(A) \subseteq C$, $f(B) \subseteq D$, and $f(\overline{A \cup B}) \subseteq \overline{C \cup D}$.[3]

THEOREM 6.7. *There exists an oracle O_2 relative to which the following holds:*

(i) *There exist $\leq_{sm}^{pp}$-complete disjoint* NP*-pairs.*

(ii) *There exist nonsymmetric disjoint* NP*-pairs.*

(iii) $\mathrm{NP} \cap \mathrm{SPARSE}$ *does not have $\leq_m^p$-complete sets.*

(iv) *Conjecture* 2.4 *holds.*

[3] $(A, B) \leq_m^{pp,X} (C, D)$ if for every separator $T \in Sep(C, D)$, there exists a separator $S \in Sep(A, B)$ such that $S \leq_m^{p,X} T$. However, since Theorems 2.10 and 2.14 hold relative to all oracles, $(A, B) \leq_m^{pp,X} (C, D)$ if and only if there is a function $f \in \mathrm{PF}^X$ such that $f(A) \subseteq C$ and $f(B) \subseteq D$. It follows immediately that $(A, B) \leq_{sm}^{pp,X} (C, D)$ implies $(A, B) \leq_m^{pp,X} (C, D)$.

Proof. In our construction we use the following witness languages, which depend on an oracle Z:

$$A(Z) \stackrel{df}{=} \{w \mid w = 0^n 10^t 1x \text{ for } n,t \geq 1,\ x \in \Sigma^* \text{ and } (\exists y \in \Sigma^{3|w|+3})[0wy \in Z]\},$$
$$B(Z) \stackrel{df}{=} \{w \mid w = 0^n 10^t 1x \text{ for } n,t \geq 1,\ x \in \Sigma^* \text{ and } (\exists y \in \Sigma^{3|w|+3})[1wy \in Z]\},$$
$$C(Z) \stackrel{df}{=} \{0^k \mid k \equiv 1 \pmod 4,\ (\exists y \in \Sigma^{k-1})[0y \in Z]\},$$
$$D(Z) \stackrel{df}{=} \{0^k \mid k \equiv 1 \pmod 4,\ (\exists y \in \Sigma^{k-1})[1y \in Z]\},$$
$$E_i(Z) \stackrel{df}{=} \{0^i 1x \mid |0^i 1x| \equiv 1 \pmod 4 \text{ and } (\exists y \in \Sigma^*,\ |y| = |0^i 1x|)[0^i 1xy \in Z]\} \quad \text{for } i \geq 1,$$
$$F(Z) \stackrel{df}{=} \{0^k \mid k \equiv 3 \pmod 4,\ (\exists y \in \Sigma^k)[y \in Z]\}.$$

These languages are in NP^Z. By definition, $A(Z)$ and $B(Z)$ depend on oracle words of length $\equiv 0 \pmod 4$, $C(Z)$ and $D(Z)$ depend on oracle words of length $\equiv 1 \pmod 4$, all $E_i(Z)$ depend on oracle words of length $\equiv 2 \pmod 4$, and $F(Z)$ depends on oracle words of length $\equiv 3 \pmod 4$. We construct the oracle O_2 such that $A(O_2) \cap B(O_2) = C(O_2) \cap D(O_2) = \emptyset$ and the following holds:

- $(A(O_2), B(O_2))$ is $\leq^{pp}_{sm}$-complete. That is,

$$\begin{aligned}&(\forall (G,H) \in \mathrm{DisjNP}^{O_2})(\exists f \in \mathrm{PF})\\ &[f(G) \subseteq A(O_2) \wedge f(H) \subseteq B(O_2) \wedge f(\overline{G \cup H}) \subseteq \overline{A(O_2) \cup B(O_2)}].\end{aligned} \tag{11}$$

- $(C(O_2), D(O_2))$ is nonsymmetric. That is,

$$(\forall f \in \mathrm{PF}^{O_2})[f(C(O_2)) \not\subseteq D(O_2) \vee f(D(O_2)) \not\subseteq C(O_2)]. \tag{12}$$

- $\mathrm{NP}^{O_2} \cap \mathrm{SPARSE}$ does not have $\leq^{p,O_2}_m$-complete sets. That is,

$$\begin{aligned}&(\forall j,\ L(NM_j^{O_2}) \in \mathrm{SPARSE}_j)(\exists n,\ E_n(O_2) \text{ contains } \leq 2 \text{ words of every length})\\ &(\forall f \in \mathrm{PF}^{O_2})[E_n(O_2) \text{ does not } \leq^{p,O_2}_m\text{-reduce to } L(NM_j^{O_2}) \text{ via } f].\end{aligned} \tag{13}$$

- $F(O_2) \not\leq^{pp,O_2}_T (A(O_2), B(O_2))$. That is,

$$(\exists S,\ A(O_2) \subseteq S \subseteq \overline{B(O_2)})[F(O_2) \notin \mathrm{P}^S]. \tag{14}$$

In (11) and (14) we really mean $f \in \mathrm{PF}$ and $F(O_2) \notin \mathrm{P}^S$; we explain why this is equivalent to $f \in \mathrm{PF}^{O_2}$ and $F(O_2) \notin \mathrm{P}^{S,O_2}$. We have to see that expressions (11), (12), (13), and (14) imply statements (i), (ii), (iii), and (iv) of Theorem 6.7. For (11) and (12) this follows from the fact that $f \in \mathrm{PF}$ implies $f \in \mathrm{PF}^{O_2}$. Each language in NP is accepted by infinitely many machines NM_j. Therefore, if there exists a sparse language L such that L is many-one-complete for $\mathrm{NP}^{O_2} \cap \mathrm{SPARSE}$, then there exists a $j \geq 1$ such that $L = L(NM_j^{O_2})$ and $L \in \mathrm{SPARSE}_j$. This shows that expression (13) implies (iii). In (14) we actually should have $F(O_2) \notin \mathrm{P}^{S,O_2}$ since the reducing machine has access to the oracle O_2. However, since (i) holds and since $(O_2, \overline{O_2}) \in \mathrm{DisjNP}^{O_2}$, there exists an $f \in \mathrm{PF}$ with $f(O_2) \subseteq A(O_2) \subseteq S$ and $f(\overline{O_2}) \subseteq B(O_2) \subseteq \overline{S}$. Hence, $q \in O_2 \Leftrightarrow f(q) \in S$. So we can transform queries to O_2 into queries to S; i.e., it suffices to show $F(O_2) \notin \mathrm{P}^S$. By expression (14), the complete pair $(A(O_2), B(O_2))$ is not NP^{O_2}-hard; it follows that no disjoint NP^{O_2}-pair is NP^{O_2}-hard.

We define the following list $\mathcal{T}$ of requirements. At the beginning of the construction, $\mathcal{T}$ contains all pairs (i,n) with $i \in \{1,2,3,4\}$ and $n \in \mathbb{N}^+$. These pairs have the following interpretations, which correspond to statements (i)–(iv) of Theorem 6.7:

- $(1, \langle i, j\rangle)$: ensure

$$L(NM_i^{O_2}) \cap L(NM_j^{O_2}) \neq \emptyset$$

or

$$(L(NM_i^{O_2}), L(NM_j^{O_2})) \leq_{sm}^{pp} (A(O_2), B(O_2)).$$

- $(2, i)$: ensure that there exists some n such that $[0^n \in C(O_2) \wedge T_i^{O_2}(0^n) \notin D(O_2)]$ or $[0^n \in D(O_2) \wedge T_i^{O_2}(0^n) \notin C(O_2)]$.
- $(3, \langle i, j\rangle)$: ensure either $L(NM_j^{O_2}) \notin \mathrm{SPARSE}_j$ or [for some n, $E_n(O_2)$ contains ≤ 2 words of every length, and $E_n(O_2)$ does not $\leq_m^{p,O_2}$-reduce to $L(NM_j^{O_2})$ via $f_i^{O_2}$] (in the construction, n does not depend on i; i.e., $(3, \langle i, j\rangle)$ and $(3, \langle i', j\rangle)$ use the same n).
- $(4, i)$: ensure that $(A(O_2), B(O_2))$ has a separator S such that $0^n \in F(O_2) \Leftrightarrow 0^n \notin L(M_i^S)$.

Once a requirement is satisfied, we delete it from the list. Conditions of the form $(2, \cdot)$ and $(4, \cdot)$ are reachable by the construction of one counterexample. In contrast, if we cannot reach $L(NM_i^{O_2}) \cap L(NM_j^{O_2}) \neq \emptyset$ for a condition of the first type, then we have to ensure $(L(NM_i^{O_2}), L(NM_j^{O_2})) \leq_{sm}^{pp} (A(O_2), B(O_2))$. Similarly, if we cannot reach $L(NM_j^{O_2}) \notin \mathrm{SPARSE}_j$ for a condition of the third type, then, for a suitable n, we have to ensure that $E_n(O_2)$ contains ≤ 2 words of every length. But these conditions cannot be reached by a finite segment of an oracle; instead they influence the whole remaining construction of the oracle. We have to encode answers to queries of the form "does x belong to $L(NM_i^{O_2})$ or to $L(NM_j^{O_2})$" into the oracle O_2, and we have to keep an eye on the number of elements of $E_n(O_2)$. For this reason we introduce the notion of (μ, k)-valid oracles. Here k is a natural number and μ is an injective, partial function $\mathbb{N}^+ \to \mathbb{N} \times \mathbb{N}^+$ that has a finite domain. Each (μ, k)-valid oracle is a subset of $\Sigma^{\leq k}$. If a pair $(0, j)$, $j \geq 1$, is in the range of μ, then this means that $L(NM_j^{O_2}) \in \mathrm{SPARSE}_j$ is forced, and therefore we must construct O_2 so that for a suitable n, $E_n(O_2)$ contains ≤ 2 words of every length. If a pair (i, j), $i, j \geq 1$, is in the range of μ, then $L(NM_i^{O_2}) \cap L(NM_j^{O_2}) = \emptyset$ is forced, and therefore we must construct O_2 so that $(L(NM_i^{O_2}), L(NM_j^{O_2})) \leq_{sm}^{pp} (A(O_2), B(O_2))$ holds. For the latter condition we have to encode certain information into O_2, and the number k says up to which level this encoding has been done. So (μ, k)-valid oracles should be considered as finite prefixes of oracles that contain these encodings. For the moment we postpone the formal definition of (μ, k)-valid oracles (Definition 6.9); instead we mention its essential properties, which we will prove later.

(a) The oracle $\emptyset$ is $(\emptyset, 0)$-valid.
(b) If X is a finite oracle that is (μ, k)-valid, then for all $\mu' \preceq \mu$, X is (μ', k)-valid.
(c) If O_2 is an oracle such that for some μ, ${O_2}^{\leq k}$ is (μ, k)-valid for infinitely many k, then the following hold:
   - $A(O_2) \cap B(O_2) = C(O_2) \cap D(O_2) = \emptyset$.
   - For all $(i, j) \in \mathrm{range}(\mu)$, if $i > 0$, then

$$(L(NM_i^{O_2}), L(NM_j^{O_2})) \leq_{sm}^{pp} (A(O_2), B(O_2))$$

     via some $f \in \mathrm{PF}$.
   - For all $(n, 0, j) \in \mu$ it holds that $E_n(O_2)$ contains ≤ 2 words of every length and $L(NM_j^{O_2}) \in \mathrm{SPARSE}_j$.

Properties (a), (b), and (c) will be proved later in Propositions 6.10 and 6.11. Moreover, we will prove the following for all $i, j \geq 1$ and all (μ, k)-valid X. (Note that there is a correspondence between (i)–(iv) and P1–P4.)

P1: There exists an $l > k$ and a (μ', l)-valid $Y \supseteq_k X$, $\mu \preceq \mu'$ such that
- either for all $Z \supseteq_l Y$, $L(NM_i^Z) \cap L(NM_j^Z) \neq \emptyset$, or
- $(i, j) \in \text{range}(\mu')$.[4]

P2: There exists an $l > k$ and a (μ, l)-valid $Y \supseteq_k X$ such that for all $Z \supseteq_l Y$, if $C(Z) \cap D(Z) = \emptyset$, then $(C(Z), D(Z))$ does not $\leq_m^{pp,O_2}$-reduce to $(D(Z), C(Z))$ via T_i^Z.

P3: (a) There exists an $l > k$ and a (μ', l)-valid $Y \supseteq_k X$, $\mu \preceq \mu'$, such that
- either for all $Z \supseteq_l Y$, $L(NM_j^Z) \notin \text{SPARSE}_j$, or
- $(0, j) \in \text{range}(\mu')$.

(b) For every n, if $\mu(n) = (0, j)$, then there exists an $l > k$ and a (μ, l)-valid $Y \supseteq_k X$ such that for all $Z \supseteq_l Y$, $E_n(Z)$ does not $\leq_m^{p,Z}$-reduce to $L(NM_j^Z)$ via f_i^Z.

P4: There exists an $l > k$ and a (μ, l)-valid $Y \supseteq_k X$ such that for all $Z \supseteq_l Y$, if $A(Z) \cap B(Z) = \emptyset$, then there exists a separator S of $(A(Z), B(Z))$ such that $F(Z) \neq L(M_i^S)$.

We will prove properties P1, P2, P3(a), P3(b), and P4 in Propositions 6.21, 6.22, 6.23, 6.25, and 6.32, respectively.

We construct an ascending sequence of finite oracles $X_0 \subseteq_{k_0} X_1 \subseteq_{k_1} X_2 \subseteq_{k_2} \cdots$ such that each X_r is (μ_r, k_r)-valid, $k_0 < k_1 < k_2 < \cdots$, and $\mu_0 \preceq \mu_1 \preceq \mu_2 \preceq \cdots$. By definition, $O_2 = \bigcup_{r \geq 0} X_r$. By items (b) and (c), $A(O_2) \cap B(O_2) = C(O_2) \cap D(O_2) = \emptyset$ follows immediately. Note that for each $r \geq 0$ and $i \geq 1$ it holds that $X_{r+i} \supseteq_{k_r} X_r$ and $\mu_r \preceq \mu_{r+i}$.

1. $r := 0$, $k_r := 0$, $\mu_r := \emptyset$, and $X_r := \emptyset$. Then by (a), X_r is (μ_r, k_r)-valid.
2. Let e be the next requirement on $\mathcal{T}$.
   (a) If $e = (1, \langle i, j \rangle)$, then we apply property P1 to X_r. Define $k_{r+1} = l$, $\mu_{r+1} = \mu'$, and $X_{r+1} = Y$. Then $k_r < k_{r+1}$, $\mu_r \preceq \mu_{r+1}$, and $X_{r+1} \supseteq_{k_r} X_r$ is (μ_{r+1}, k_{r+1})-valid such that
   - either for all $Z \supseteq_{k_{r+1}} X_{r+1}$, $L(NM_i^Z) \cap L(NM_j^Z) \neq \emptyset$, or
   - $(i, j) \in \text{range}(\mu_{r+1})$.

   Remove e from $\mathcal{T}$ and go to step 3.

   Comment: If the former holds, then, since $O_2 \supseteq_{k_{r+1}} X_{r+1}$, it holds that $L(NM_i^{O_2}) \cap L(NM_j^{O_2}) \neq \emptyset$, and therefore $(L(NM_i^{O_2}), L(NM_j^{O_2})) \notin \text{DisjNP}^{O_2}$. Otherwise, $(i, j) \in \text{range}(\mu_{r+1})$. By (b), for all $i \geq 1$, X_{r+i} is (μ_{r+1}, k_{r+i})-valid. Therefore, by (c), $(L(NM_i^{O_2}), L(NM_j^{O_2})) \leq_{sm}^{pp} (A(O_2), B(O_2))$ via some $f \in \text{PF}$.

   (b) If $e = (2, i)$, then $\mu_{r+1} \stackrel{df}{=} \mu_r$ and apply property P2 to X_r. We define $k_{r+1} = l$ and $X_{r+1} = Y$. Then $k_{r+1} > k_r$ and $X_{r+1} \supseteq_{k_r} X_r$ is (μ_{r+1}, k_{r+1})-valid so that for all $Z \supseteq_{k_{r+1}} X_{r+1}$, if $C(Z) \cap D(Z) = \emptyset$, then $(C(Z), D(Z))$ does not $\leq_m^{pp,O_2}$-reduce to $(D(Z), C(Z))$ via T_i^Z. Remove e from $\mathcal{T}$ and go to step 3.

   Comment: Since $O_2 \supseteq_{k_{r+1}} X_{r+1}$ and $C(O_2) \cap D(O_2) = \emptyset$, this ensures that $(C(O_2), D(O_2))$ does not $\leq_m^{pp,O_2}$-reduce to $(D(O_2), C(O_2))$ via $T_i^{O_2}$.

   (c) If $e = (3, \langle i, j \rangle)$ and $(0, j) \notin \text{range}(\mu_r)$, then we apply property P3(a) to X_r. Define $k_{r+1} = l$, $\mu_{r+1} = \mu'$, and $X_{r+1} = Y$. Then $k_r < k_{r+1}$, $\mu_r \preceq \mu_{r+1}$, and $X_{r+1} \supseteq_{k_r} X_r$ is (μ_{r+1}, k_{r+1})-valid such that
   - either for all $Z \supseteq_{k_{r+1}} X_{r+1}$, $L(NM_j^Z) \notin \text{SPARSE}_j$, or

[4] Proposition 6.21 says $L(NM_i^Z) \cap L(NM_j^Z) \cap \Sigma^{\leq l} \neq \emptyset$, which is a stronger statement.

- $(0, j) \in \text{range}(\mu_{r+1})$.

If the former holds, then remove e from $\mathcal{T}$ and go to step 3. Otherwise, do not remove e from $\mathcal{T}$ (it will be removed in the next iteration) and go to step 3.

Comment: If the former of the two alternatives holds, then, since $O_2 \supseteq_{k_{r+1}} X_{r+1}$, it holds that $L(NM_j^{O_2}) \notin \text{SPARSE}_j$. Otherwise, for a suitable n, $(n, 0, j) \in \mu_{r+1}$. By (b), for all $i \geq 1$, X_{r+i} is (μ_{r+1}, k_{r+i})-valid. Therefore, by (c), it is enforced that $E_n(O_2)$ contains ≤ 2 words of every length and $L(NM_j^{O_2}) \in \text{SPARSE}_j$. From now on, all requirements of the form $(3, \langle \cdot, j \rangle)$ are treated in step 2(d). These steps will make sure that $E_n(O_2) \not\leq_m^{p,O_2} L(NM_j^{O_2})$.

(d) If $e = (3, \langle i, j \rangle)$ and $(0, j) \in \text{range}(\mu_r)$, then choose n such that $(n, 0, j) \in \mu_r$ and apply property P3(b) to X_r. Define $k_{r+1} = l$, $\mu_{r+1} = \mu_r$, and $X_{r+1} = Y$. Then $k_r < k_{r+1}$, $\mu_r \preceq \mu_{r+1}$, and $X_{r+1} \supseteq_{k_r} X_r$ is (μ_{r+1}, k_{r+1})-valid such that for all $Z \supseteq_{k_{r+1}} X_{r+1}$, $E_n(Z)$ does not $\leq_m^{p,Z}$-reduce to $L(NM_j^Z)$ via f_i^Z. Remove e from $\mathcal{T}$ and go to step 3.

Comment: In the comment of the previous step we have seen that $(0, j) \in \text{range}(\mu_r)$ implies that $E_n(O_2) \in \text{SPARSE}_{j+1}$. Since $O_2 \supseteq_{k_{r+1}} X_{r+1}$ this step ensures that $E_n(O_2)$ does not $\leq_m^{p,O_2}$-reduce to $L(NM_j^{O_2})$ via $f_i^{O_2}$.

(e) If $e = (4, i)$, then $\mu_{r+1} \stackrel{df}{=} \mu_r$ and apply property P4 to X_r. We define $k_{r+1} = l$ and $X_{r+1} = Y$. Then $k_{r+1} > k_r$ and $X_{r+1} \supseteq_{k_r} X_r$ is (μ_{r+1}, k_{r+1})-valid such that for all $Z \supseteq_{k_{r+1}} X_{r+1}$, if $A(Z) \cap B(Z) = \emptyset$, then there exists a separator S of $(A(Z), B(Z))$ such that $F(Z) \neq L(M_i^S)$. Remove e from $\mathcal{T}$ and go to step 3.

Comment: Since $O_2 \supseteq_{k_{r+1}} X_{r+1}$ and $A(O_2) \cap B(O_2) = \emptyset$, this ensures that there exists a separator S of $(A(O_2), B(O_2))$ such that $F(O_2) \neq L(M_i^S)$.

3. $r := r + 1$, go to step 2.

We see that this construction ensures (i), (ii), (iii), and (iv). This proves Theorem 6.7 except to show that we can define an appropriate notion of a (μ, k)-valid oracle that has properties (a), (b), (c) and P1, P2, P3, P4.

We want to construct our oracle such that $(A(O_2), B(O_2))$ is a $\leq_{sm}^{pp}$-complete disjoint NP^{O_2}-pair. So we have to make sure that pairs $(L(NM_i), L(NM_j))$ that are enforced to be disjoint (which means that $(i, j) \in \text{range}(\mu)$) can be $\leq_{sm}^{pp}$-reduced to $(A(O_2), B(O_2))$. Therefore, we put certain codewords into O_2 if and only if the computation $NM_i^{O_2}(x)$ (resp., $NM_j^{O_2}(x)$) accepts within t steps.

DEFINITION 6.8 (μ-codeword). *Let $\mu : \mathbb{N}^+ \to \mathbb{N} \times \mathbb{N}^+$ be an injective, partial function with a finite domain. A word w is called a* μ-codeword *if $w = 00^n10^t1xy$ or $w = 10^n10^t1xy$ such that $n, t \geq 1$, $|y| = 3|00^n10^t1x|$, and $\mu(n) = (i, j)$ such that $i, j \geq 1$. If $w = 00^n10^t1xy$, then we say that w is a μ-codeword for (i, t, x); if $w = 10^n10^t1xy$, then we say it is a μ-codeword for (j, t, x).*

Condition (i) of Theorem 6.7 opposes conditions (ii), (iii), and (iv), because for (i) we have to encode information about NP^{O_2} computations into O_2, and (ii), (iii), and (iv) say that we cannot encode too much information (e.g., enough information for $\text{UP}^{O_2} = \text{NP}^{O_2}$). For this reason we have to look at certain finite oracles that contain the needed information for (i) and that allow all diagonalization needed to reach (ii), (iii), and (iv). We call such oracles (μ, k)-valid.

DEFINITION 6.9 ((μ, k)-valid oracle). *Let $k \geq 0$ and let $\mu : \mathbb{N}^+ \to \mathbb{N} \times \mathbb{N}^+$ be an injective, partial function with a finite domain. We define a finite oracle X to be (μ, k)-valid by induction over the size of the domain of μ.*

(IB) *If $\|\mu\| = 0$, then X is (μ, k)-valid $\stackrel{df}{\iff}$ $X \subseteq \Sigma^{\leq k}$ and $A(X) \cap B(X) = C(X) \cap D(X) = \emptyset$.*

(IS) *If* $\|\mu\| > 0$, *then* $\mu = \mu_0 \cup \{(n_0, i_0, j_0)\}$, *where* $n_0 = \mu_{\max}$ *and* $\mu_0 \prec \mu$. X *is* (μ, k)*-valid* $\overset{df}{\iff}$ $k \geq n_0$, X *is* (μ_0, k)*-valid, and the following holds:*

1. *If* $i_0 > 0$, *then we demand the following:*
   (a) *For all* $t \geq 1$ *and all* $x \in \Sigma^*$, *if* $4 \cdot |00^{n_0}10^t1x| \leq k$, *then*
      (i) $(\exists y,\ |y| = 3|00^{n_0}10^t1x|)[00^{n_0}10^t1xy \in X] \Leftrightarrow NM_{i_0}^X(x)$ *accepts within* t *steps, and*
      (ii) $(\exists y,\ |y| = 3|10^{n_0}10^t1x|)[10^{n_0}10^t1xy \in X] \Leftrightarrow NM_{j_0}^X(x)$ *accepts within* t *steps.*
   (b) *For all* $l \geq n_0$ *and all* (μ_0, l)*-valid* Y, *if* $Y^{\leq n_0} = X^{\leq n_0}$, *then* $L(NM_{i_0}^Y) \cap L(NM_{j_0}^Y) \cap \Sigma^{\leq l} = \emptyset$.
2. *If* $i_0 = 0$, *then*
   (a) *for every* $r \geq 0$, $\|E_{n_0}(X) \cap \Sigma^r\| \leq 2$, *and*
   (b) *for all* $l \geq n_0$ *and all* (μ_0, l)*-valid* Y, *if* $Y^{\leq n_0} = X^{\leq n_0}$, *then* $L(NM_{j_0}^Y) \cap \Sigma^{\leq l} \in \mathrm{SPARSE}_{j_0}$.

Due to conditions 1(b) and 2(b), (μ, k)-valid oracles can be extended to (μ, k')-valid oracles with $k' > k$ (Lemma 6.17). There we really need the intersection with $\Sigma^{\leq l}$. Otherwise—for example, in 1(b)—it could be possible that for a small oracle $Y \subseteq \Sigma^{\leq l}$ both machines accept the same word w that is much longer than l, but there is no way to extend Y in a valid way to the level $|w|$ such that both machines still accept w (the reason is that the reservations (Definition 6.12) become too large).

Proposition 6.10 (basic properties of validity).

1. *The oracle* $\emptyset$ *is* $(\emptyset, 0)$*-valid (property* (a)*).*
2. *For every* (μ, k)*-valid* X *and every* $\mu' \preceq \mu$, X *is* (μ', k)*-valid (property* (b)*).*
3. *For every* (μ, k)*-valid* X *and every* $(n, 0, j) \in \mu$, *it holds that*
   (a) *for every* $r \geq 0$, $\|E_n(X) \cap \Sigma^r\| \leq 2$, *and*
   (b) $L(NM_j^X) \cap \Sigma^{\leq k} \in \mathrm{SPARSE}_j$.
4. *Let* X *be* (μ, k)*-valid and* $S \subseteq \Sigma^{k+1}$ *such that* $k + 1 \not\equiv 0 \pmod 4$, $C(S) \cap D(S) = \emptyset$, *and for all* $(n, 0, j) \in \mu$ *it holds that* $\|E_n(S)\| \leq 2$. *Then* $X \cup S$ *is* $(\mu, k+1)$*-valid.*
5. *For every* (μ, k)*-valid* X *and every* $(i, j) \in \mathrm{range}(\mu)$, $i > 0$, *it holds that* $L(NM_i^X) \cap L(NM_j^X) \cap \Sigma^{\leq k} = \emptyset$.
6. *If* X *is* (μ, k)*-valid, then for every* k', $\mu_{\max} \leq k' \leq k$ *(resp.,* $0 \leq k' \leq k$ *if* $\mu = \emptyset$*), it holds that* $X^{\leq k'}$ *is* (μ, k')*-valid.*

Proof. Statements 6.10.1 and 6.10.2 follow immediately from Definition 6.9.

Let X be (μ, k)-valid and let $(n, 0, j) \in \mu$. Let $n_0 \overset{df}{=} n$, $i_0 \overset{df}{=} 0$, $j_0 \overset{df}{=} j$, and $\mu_0 \overset{df}{=} \{(n', i', j') \in \mu \mid n' < n\}$. By 6.10.2, X is $(\mu_0 \cup \{(n_0, i_0, j_0)\}, k)$-valid and also (μ_0, k)-valid. From 6.9.2(a) it follows that 6.10.3(a) holds. From 6.9.2(b) (for $l = k$ and $Y = X$) we obtain $L(NM_{j_0}^X) \cap \Sigma^{\leq k} \in \mathrm{SPARSE}_{j_0}$. This shows 6.10.3(b).

We prove statement 6.10.4 by induction on $\|\mu\|$. First of all we see that $A(S) = B(S) = \emptyset$, since S contains no words of length $\equiv 0 \pmod 4$. If $\|\mu\| = 0$, then, by Definition 6.9, $X \cup S$ is $(\mu, k+1)$-valid. So assume $\|\mu\| > 0$ and choose μ_0, n_0, i_0, j_0 as in Definition 6.9. We assume as an induction hypothesis that if X is (μ_0, k)-valid, then $X \cup S$ is $(\mu_0, k+1)$-valid. We verify Definition 6.9 for $X \cup S$ and $k+1$. Clearly, $k + 1 > k \geq n_0$. Since X is (μ, k)-valid it is also (μ_0, k)-valid. By the induction hypothesis we obtain that $X \cup S$ is $(\mu_0, k+1)$-valid.

Assume that $i_0 > 0$; we verify item 1 of Definition 6.9. Since $k + 1 \not\equiv 0 \pmod 4$, the condition $4 \cdot |00^{n_0}10^t1x| \leq k + 1$ is equivalent to $4 \cdot |00^{n_0}10^t1x| \leq k$. Since $t < k$, the computations mentioned in 6.9.1(a) cannot ask queries longer than k. So nothing changes when these machines use oracle X instead of $X \cup S$. Moreover, at the left-

hand sides in 6.9.1(a), we can also use X instead of $X \cup S$ since we only test the membership for words of length $\equiv 0 \pmod 4$. This shows that in 6.9.1(a) we can replace every occurrence of $X \cup S$ with X and obtain an equivalent condition. This condition holds since X is (μ, k)-valid. Therefore, 6.9.1(a) holds for $X \cup S$ and $k+1$. Condition 6.9.1(b) holds for $X \cup S$ and $k+1$, since this condition does not depend on k and since $(X \cup S) \cap \Sigma^{\leq k} = X^{\leq k}$.

Assume that $i_0 = 0$; we verify item 2 of Definition 6.9. By assumption, $\|E_{n_0}(S)\| \leq 2$ and (since X is (μ, k)-valid) for all $r \geq 0$, it holds that $\|E_{n_0}(X) \cap \Sigma^r\| \leq 2$. Words in $E_{n_0}(X)$ are of length $\leq \lfloor k/2 \rfloor$. In contrast, words in $E_{n_0}(S)$ are of length $\lceil (k+1)/2 \rceil$. Hence, words in $E_{n_0}(X)$ are shorter than words in $E_{n_0}(S)$. So for all $r \geq 0$,

$$\begin{aligned}\|E_{n_0}(X \cup S) \cap \Sigma^r\| &= \|(E_{n_0}(X) \cap \Sigma^r) \cup (E_{n_0}(S) \cap \Sigma^r)\| \\ &= \|(E_{n_0}(X) \cap \Sigma^r)\| + \|(E_{n_0}(S) \cap \Sigma^r)\| \leq 2.\end{aligned}$$

This shows 6.9.2(a). Condition 6.9.2(b) holds for $X \cup S$ and $k+1$, since this condition does not depend on k, and since $(X \cup S) \cap \Sigma^{\leq k} = X^{\leq k}$. This proves statement 6.10.4.

We prove statement 5 of Proposition 6.10 as follows. Let X be (μ, k)-valid and $(i_0, j_0) \in \text{range}(\mu)$ such that $i_0 > 0$. Choose n_0 such that $(n_0, i_0, j_0) \in \mu$. Let $\mu_0 \stackrel{df}{=} \{(n', i', j') \in \mu \mid n' < n_0\}$. By 6.10.2, X is $(\mu_0 \cup \{(n_0, i_0, j_0)\}, k)$-valid and also (μ_0, k)-valid. Together with 6.9.1(b) (for $l = k$ and $Y = X$) this implies that $L(NM_{i_0}^X) \cap L(NM_{j_0}^X) \cap \Sigma^{\leq k} = \emptyset$.

We prove statement 6 of Proposition 6.10 by induction on $\|\mu\|$. If $\|\mu\| = 0$, then, by Definition 6.9, $X^{\leq k'}$ is (μ, k')-valid for $0 \leq k' \leq k$. So assume $\|\mu\| > 0$ and choose μ_0, n_0, i_0, j_0 as in Definition 6.9. We assume as an induction hypothesis that if X is (μ_0, k)-valid, then, for every k', $n_0 \leq k' \leq k$, it holds that $X^{\leq k'}$ is (μ_0, k')-valid. Choose k' such that $n_0 \leq k' \leq k$; we show that $X^{\leq k'}$ is (μ, k')-valid. Since X is (μ, k)-valid it is also (μ_0, k)-valid. By the induction hypothesis we obtain that $X^{\leq k'}$ is (μ_0, k')-valid.

Assume that $i_0 > 0$; we verify Definition 6.9.1. Note that in 6.9.1(a) we have the condition $4 \cdot |00^{n_0}10^t1x| \leq k'$. Hence, $t < k'$, and therefore the computations mentioned in 6.9.1(a) cannot ask queries longer than k'. So nothing changes when these machines use oracle X instead of $X^{\leq k'}$. Moreover, at the left-hand sides in 6.9.1(a), we can also use X instead of $X^{\leq k'}$ since we only test the membership for words of length $\leq k'$. This shows that in 6.9.1(a) we can replace every occurrence of $X^{\leq k'}$ with X and obtain an equivalent condition. This condition holds since X is (μ, k)-valid. Therefore, 6.9.1(a) holds. Condition 6.9.1(b) holds, since $X^{\leq k'} \cap \Sigma^{\leq n_0} = X^{\leq n_0}$.

Assume that $i_0 = 0$; we verify Definition 6.9.2. Condition 6.9.2(a) follows immediately, since X is (μ, k)-valid. Condition 6.9.2(b) holds, since $X^{\leq k'} \cap \Sigma^{\leq n_0} = X^{\leq n_0}$. This proves statement 6 of Proposition 6.10. □

PROPOSITION 6.11. *Let O_2 be an oracle such that for some μ there exist infinitely many k such that $O_2{}^{\leq k}$ is (μ, k)-valid (property* (c)*).*

1. $A(O_2) \cap B(O_2) = C(O_2) \cap D(O_2) = \emptyset$.
2. *For all $(i, j) \in \text{range}(\mu)$, $i > 0$, it holds that $L(NM_i^{O_2}) \cap L(NM_j^{O_2}) = \emptyset$ and there exists some $f \in \text{PF}$ such that $(L(NM_i^{O_2}), L(NM_j^{O_2})) \leq_{sm}^{pp} (A(O_2), B(O_2))$ via f.*
3. *For all $(n, 0, j) \in \mu$ it holds that $E_n(O_2)$ contains ≤ 2 words of every length, and $L(NM_j^{O_2}) \in \text{SPARSE}_j$.*

Proof. Assume that $A(O_2) \cap B(O_2) \neq \emptyset$ and let $w \in A(O_2) \cap B(O_2)$. Then, for $k = 4 \cdot (|w| + 1)$, w is already in $A(O_2{}^{\leq k}) \cap B(O_2{}^{\leq k})$. This contradicts the

assumption that there exists a $k' \geq k$ such that ${O_2}^{\leq k'}$ is (μ, k')-valid. Therefore, $A(O_2) \cap B(O_2) = \emptyset$. Analogously we see that $C(O_2) \cap D(O_2) = \emptyset$. This shows item 1 of Proposition 6.11.

Let $(i,j) \in \text{range}(\mu)$, $i > 0$, and choose n such that $(n,i,j) \in \mu$. Assume $L(NM_i^{O_2}) \cap L(NM_j^{O_2}) \neq \emptyset$, and let $w \in L(NM_i^{O_2}) \cap L(NM_j^{O_2})$. Then, for $k = |w|^{i+j}$, w is already in $L(NM_i^{O_2'}) \cap L(NM_j^{O_2'}) \cap \Sigma^{\leq k}$, where ${O_2}' \stackrel{df}{=} {O_2}^{\leq k}$. By our assumption there exists a $k' \geq k$ such that ${O_2}'' \stackrel{df}{=} {O_2}^{\leq k'}$ is (μ, k')-valid. It follows that $w \in L(NM_i^{O_2''}) \cap L(NM_j^{O_2''}) \cap \Sigma^{\leq k'}$. This contradicts Proposition 6.10.5, and therefore $L(NM_i^{O_2}) \cap L(NM_j^{O_2}) = \emptyset$.

Let $\mu_0 \stackrel{df}{=} \{(n',i',j') \in \mu \mid n' < n\}$. From our assumption and Proposition 6.10.2 it follows that for infinitely many k, ${O_2}^{\leq k}$ is $(\mu_0 \cup \{(n,i,j)\}, k)$-valid. So by Definition 6.9, for infinitely many k the following holds: For all $t \geq 1$ and all $x \in \Sigma^*$, if $4 \cdot |00^n10^t1x| \leq k$, then

- $(\exists y,\ |y| = 3|00^n10^t1x|)[00^n10^t1xy \in {O_2}^{\leq k}] \Leftrightarrow NM_i^{{O_2}^{\leq k}}(x)$ accepts within t steps, and
- $(\exists y,\ |y| = 3|10^n10^t1x|)[10^n10^t1xy \in {O_2}^{\leq k}] \Leftrightarrow NM_j^{{O_2}^{\leq k}}(x)$ accepts within t steps.

During the first t steps a machine can ask queries of length $\leq t < k$ only. Therefore, above we can replace $NM_i^{{O_2}^{\leq k}}(x)$ and $NM_j^{{O_2}^{\leq k}}(x)$ by $NM_i^{O_2}(x)$ and $NM_j^{O_2}(x)$, respectively. Moreover, since we have the condition $4 \cdot |00^n10^t1x| \leq k$, we can replace ${O_2}^{\leq k}$ with O_2 on the left-hand sides. Since the resulting condition holds for infinitely many k, the following holds for all $t \geq 1$ and $x \in \Sigma^*$:

- $(\exists y,\ |y| = 3|00^n10^t1x|)[00^n10^t1xy \in O_2] \Leftrightarrow NM_i^{O_2}(x)$ accepts within t steps.
- $(\exists y,\ |y| = 3|10^n10^t1x|)[10^n10^t1xy \in O_2] \Leftrightarrow NM_j^{O_2}(x)$ accepts within t steps.

The left-hand sides of these equivalences say $0^n10^t1x \in A(O_2)$ and $0^n10^t1x \in B(O_2)$, respectively. This shows that $(L(NM_i^{O_2}), L(NM_j^{O_2})) \leq_{sm}^{pp} (A(O_2), B(O_2))$ via some $f \in \text{PF}$.[5] Hence statement 2 of Proposition 6.11 holds.

Let $(n,0,j) \in \mu$. Assume that there exists an $r \geq 0$ such that $\|E_n(O_2) \cap \Sigma^r\| \geq 3$. Then there exists some k such that $\|E_n({O_2}') \cap \Sigma^r\| \geq 3$, where ${O_2}' \stackrel{df}{=} {O_2}^{\leq k}$. By our assumption there exists some $k' \geq k$ such that ${O_2}'' \stackrel{df}{=} {O_2}^{\leq k'}$ is (μ, k')-valid. It follows that $\|E_n({O_2}'') \cap \Sigma^r\| \geq 3$. This contradicts Proposition 6.10.3(a), and therefore $E_n(O_2)$ contains at most two words of every length.

Assume that $L(NM_j^{O_2}) \notin \text{SPARSE}_j$. Then there exists some m such that $L(NM_j^{O_2}) \cap \Sigma^m$ contains more than $m^j + j$ words. Therefore, with $k \stackrel{df}{=} m^j$ and ${O_2}' \stackrel{df}{=} {O_2}^{\leq k}$ we obtain $L(NM_j^{O_2'}) \cap \Sigma^{\leq k} \notin \text{SPARSE}_j$. By our assumption there exists some $k' \geq k$ such that ${O_2}'' \stackrel{df}{=} {O_2}^{\leq k'}$ is (μ, k')-valid. It follows that $L(NM_j^{O_2''}) \cap \Sigma^{\leq k'} \notin \text{SPARSE}_j$. This contradicts Proposition 6.10.3(b), and therefore $L(NM_j^{O_2}) \in \text{SPARSE}_j$. □

Remember that our construction consists of a coding part to obtain condition (i) of Theorem 6.7 and of separating parts to obtain conditions (ii), (iii), and (iv). In order to diagonalize, we will fix certain words that are needed for the coding part, and we will change our oracle on nonfixed positions to obtain the separation. For this we introduce the notion of a reservation for an oracle. A reservation consists of two sets Y and N, where Y contains words that are reserved for the oracle while N

[5] We can use $f(x) \stackrel{df}{=} 0^n10^{|x|^{i+j}}1x$, since $NM_i(x)$ and $NM_j(x)$ have computation times $|x|^i$ and $|x|^j$, respectively.

contains words that are reserved for the complement of the oracle. This notion has two important properties:

- Whenever an oracle X agrees with a reservation that is not too large, we can find an extension of X that agrees with the reservation (Lemma 6.14).
- If we want to fix certain words to be in the oracle, then this is possible using a reservation of small size. For this reason we can fix certain words to be in the oracle and still be able to diagonalize (Lemma 6.18).

DEFINITION 6.12 ((μ,k)-reservation). *A pair (Y,N) of finite sets is a (μ,k)-reservation for X if X is (μ,k)-valid, $Y\cap N=\emptyset$, $Y^{\leq k}\subseteq X$, $N^{\leq k}\subseteq \overline{X}$, $A(Y)\cap B(Y)=\emptyset$, all words in $Y^{>k}$ are of length $\equiv 0 \pmod 4$, and if $w\in Y^{>k}$ is a μ-codeword for (i,t,x), then $NM_i(x)$ has a positive path P such that $|P|\leq t$, $P^{\text{yes}}\subseteq Y$, and $P^{\text{no}}\subseteq N$.*

PROPOSITION 6.13 (basic properties of reservations). *The following holds for every (μ,k)-valid X:*

1. *$(\emptyset,\emptyset)$ is a (μ,k)-reservation for X.*
2. *If (Y,N) is a (μ,k)-reservation for X, then also $(Y,N\cup N')$ for every $N'\subseteq \overline{Y\cup X}$.*
3. *For every $N\subseteq\overline{X}$, $(\emptyset,N)$ is a (μ,k)-reservation for X.*
4. *Let (Y,N) be a (μ,k)-reservation for X. For each $(\mu,k+1)$-valid $Z\supseteq_k X$ such that $Y^{=k+1}\subseteq Z^{=k+1}\subseteq \overline{N}^{=k+1}$, it holds that (Y,N) is a $(\mu,k+1)$-reservation for Z.*
5. *Let (Y,N) be a (μ,k)-reservation for X. For every $m\geq 0$, $(Y\cap\Sigma^{\leq m}, N\cap\Sigma^{\leq m})$ is a (μ,k)-reservation for X.*

Proof. This follows immediately from Definition 6.12. □

Whenever a (μ,k)-reservation of some oracle X is not too large, then X has a (μ,m)-valid extension Z that agrees with the reservation.

LEMMA 6.14. *Let (Y,N) be a (μ,k)-reservation for X and let $m\stackrel{df}{=}\max(\{|w|\mid w\in Y\cup N\}\cup\{k\})$. If $\|N\|\leq 2^{k/2}$, then there exists a (μ,m)-valid $Z\supseteq_k X$ such that $Y\subseteq Z$, $N\subseteq\overline{Z}$, and $(Z-Y)\cap\Sigma^{>k}$ contains only μ-codewords.*

Proof. Assume $\|N\|\leq 2^{k/2}$. We show the lemma by induction on $n\stackrel{df}{=}m-k$. If $n=0$, then let $Z=X$ and we are done.

Now assume $n>0$. First of all we show that it suffices to find a $(\mu,k+1)$-valid $Z'\supseteq_k X$ such that $Y^{=k+1}\subseteq Z'^{=k+1}\subseteq\overline{N}^{=k+1}$ and $(Z'-Y)\cap\Sigma^{k+1}$ contains only μ-codewords. In this case, Proposition 6.13.4 implies that (Y,N) is a $(\mu,k+1)$-reservation for Z'. So we can apply the induction hypothesis to (Y,N) considered as a $(\mu,k+1)$-reservation for Z'. We obtain a (μ,m)-valid $Z\supseteq_{k+1}Z'$ such that $Y\subseteq Z$, $N\subseteq\overline{Z}$, and $(Z-Y)\cap\Sigma^{>k+1}$ contains only μ-codewords. Together this yields $Z\supseteq_k X$ and $(Z-Y)\cap\Sigma^{>k}$ contains only μ-codewords. It remains to find the mentioned Z'.

If $k+1\not\equiv 0 \pmod 4$, then $Y^{=k+1}=\emptyset$, since $Y^{=k+1}$ contains only words of length $\equiv 0 \pmod 4$. We apply Proposition 6.10.4 to $S\stackrel{df}{=}\emptyset$, and obtain that X is $(\mu,k+1)$-valid. Therefore, with $Z'\stackrel{df}{=}X$ we found the desired Z'.

If $k+1\equiv 0 \pmod 4$, then, starting with the empty set, we construct a set $S\subseteq\Sigma^{k+1}$ by doing the following for each $(n,i,j)\in\mu$, each $t\geq 1$, and each $x\in\Sigma^*$ such that $i>0$ and $4\cdot|00^n10^t1x|=k+1$:

- If $NM_i^X(x)$ accepts within t steps, then choose some $y\in\Sigma^{3|00^n10^t1x|}$ such that $00^n10^t1xy\notin N$. Add 00^n10^t1xy to S.
- If $NM_j^X(x)$ accepts within t steps, then choose some $y\in\Sigma^{3|10^n10^t1x|}$ such that $10^n10^t1xy\notin N$. Add 10^n10^t1xy to S.

Observe that the choices of words y are possible since $\|N\| \leq 2^{k/2} < 2^{3(k+1)/4} = \|\Sigma^{3|00^n10^t1x|}\|$. Moreover, S contains only μ-codewords. For $Z' \stackrel{df}{=} X \cup S \cup Y^{=k+1}$ we have $Z' \supseteq_k X$ and $Y^{=k+1} \subseteq Z'^{=k+1} \subseteq \overline{N}^{=k+1}$, since $S \subseteq \overline{N}^{=k+1}$. In addition, $(Z' - Y) \cap \Sigma^{k+1}$ contains only μ-codewords, since this set is a subset of S. It remains to show that Z' is $(\mu, k+1)$-valid.

CLAIM 6.15. $A(Z') \cap B(Z') = C(Z') \cap D(Z') = \emptyset$.

Proof. Since X is (μ, k)-valid we have $A(X) \cap B(X) = C(X) \cap D(X) = \emptyset$. When we look at the definitions of $A(X)$, $B(X)$, $C(X)$, and $D(X)$, we see that in order to show Claim 6.15, it suffices to show

$$A(Z') \cap B(Z') \cap \Sigma^{\frac{(k+1)}{4}-1} = C(Z') \cap D(Z') \cap \Sigma^{k+1} = \emptyset.$$

We immediately obtain $C(Z') \cap D(Z') \cap \Sigma^{k+1} = \emptyset$, since by definition, $C(Z')$ and $D(Z')$ contain only words of lengths $\equiv 1 \pmod 4$. Assume that $A(Z') \cap B(Z') \cap \Sigma^{(k+1)/4-1} \neq \emptyset$, and choose some $w \in A(Z') \cap B(Z') \cap \Sigma^{(k+1)/4-1}$. So there exist $n, t \geq 1$, $x \in \Sigma^*$, and $y_0, y_1 \in \Sigma^{3|w|+3}$ such that $w = 0^n10^t1x$ and $0wy_0, 1wy_1 \in Z'$. Note that $0wy_0, 1wy_1 \in S \cup Y^{=k+1}$, but both words cannot be in $Y^{=k+1}$, since otherwise we have $A(Y) \cap B(Y) \neq \emptyset$, which contradicts our assumption that (Y, N) is a (μ, k)-reservation. Therefore, either $0wy_0$ or $1wy_1$ belongs to S. Since all words in S are μ-codewords, there exist $i, j \geq 1$ such that $(n, i, j) \in \mu$. Hence $0wy_0$ and $1wy_1$ are μ-codewords. We claim that $NM_i^X(x)$ accepts within t steps, regardless of whether $0wy_0$ belongs to S or to $Y^{=k+1}$. This can be seen as follows:

- If $0wy_0 \in S$, then from the construction of S it follows that $NM_i^X(x)$ accepts within t steps.
- If $0wy_0 \in Y^{=k+1}$, then, since $0wy_0$ is a μ-codeword of length $> k$, $NM_i(x)$ has a positive path P with $|P| \leq t$, $P^{\text{yes}} \subseteq Y$, and $P^{\text{no}} \subseteq N$. Since $t \leq k$ it follows that $P^{\text{yes}} \cup P^{\text{no}} \subseteq \Sigma^{\leq k}$, and therefore $P^{\text{yes}} \subseteq X$ and $P^{\text{no}} \subseteq \Sigma^{\leq k} - X$. It follows that $NM_i^X(x)$ accepts within t steps.

Analogously we obtain that $NM_j^X(x)$ accepts within t steps. Since $|x| \leq k$ we have seen that $L(NM_i^X) \cap L(NM_j^X) \cap \Sigma^{\leq k} \neq \emptyset$ and $(i, j) \in \text{range}(\mu)$ such that $i > 0$. This contradicts Proposition 6.10.5 and finishes the proof of Claim 6.15. □

CLAIM 6.16. *Z' is $(\mu', k+1)$-valid for every $\mu' \preceq \mu$.*

Proof. We prove the claim by induction on $\|\mu'\|$. If $\|\mu'\| = 0$, then Z' is $(\mu', k+1)$-valid by Claim 6.15.

Assume now that $\|\mu'\| > 0$, and choose suitable μ_0, n_0, i_0, j_0 such that $n_0 = \mu'_{\max}$, $\mu' = \mu_0 \cup \{(n_0, i_0, j_0)\}$, and $\mu_0 \prec \mu'$. Clearly, $n_0 \leq \mu_{\max} \leq k < k+1$. As an induction hypothesis we assume that Z' is $(\mu_0, k+1)$-valid. We show that Z' is $(\mu', k+1)$-valid.

Assume $i_0 > 0$. We claim that for all $t \geq 1$ and all $x \in \Sigma^*$, if $4 \cdot |00^{n_0}10^t1x| \leq k+1$, then the equivalences in 6.9.1(a) hold for Z' instead of X. This is seen as follows:

- If $4 \cdot |00^{n_0}10^t1x| \leq k$, then they hold since X is (μ', k)-valid and $Z' \supseteq_k X$.
- If $4 \cdot |00^{n_0}10^t1x| = k+1$, then the implications "$\Leftarrow$" in statement 6.9.1(a) hold, since $NM_{i_0}^{Z'}(x)$ and $NM_{j_0}^{Z'}(x)$ run at most $t \leq k$ steps and can therefore use oracle X instead of Z', and because $S \subseteq Z'$. For the other direction, let $w = 0^{n_0}10^t1x$ and assume that there exists some $y \in \Sigma^{3|w|+3}$ such that $0wy \in Z'$. If $0wy \in S$, then we have put this word to S, because $NM_i^X(x)$ accepts within t steps. Since $t < k$, also $NM_i^{Z'}(x)$ accepts within t steps. So assume $0wy \in Y^{=k+1}$ and note that $0wy$ is a μ-codeword. Since (Y, N) is a (μ, k)-reservation for X, $NM_i(x)$ has a positive path P with $|P| \leq t$, $P^{\text{yes}} \subseteq Y$, and $P^{\text{no}} \subseteq N$. Since $t < k$, we have $P^{\text{yes}} \subseteq X$ and $P^{\text{no}} \subseteq \Sigma^{\leq k} - X$.

Hence, $NM_i^X(x)$ accepts within t steps, and therefore $NM_i^{Z'}(x)$ accepts within t steps. This shows the implication "$\Rightarrow$" in 6.9.1(a)(i). Analogously we see the implication "$\Rightarrow$" in 6.9.1(a)(ii).

Condition 6.9.1(b) holds for Z' instead of X, since X is (μ', k)-valid, $n_0 \leq k$ and therefore $Z'^{\leq n_0} = X^{\leq n_0}$.

Assume $i_0 = 0$. Since X is (μ', k)-valid, for all $r \geq 0$ it holds that $\|E_{n_0}(X) \cap \Sigma^r\| \leq 2$. Moreover, we have $E_{n_0}(Z' \cap \Sigma^{k+1}) = \emptyset$, since by definition, E_{n_0} depends only on oracle words of lengths $\equiv 2 \pmod 4$. Therefore, for all $r \geq 0$, $\|E_{n_0}(Z') \cap \Sigma^r\| \leq 2$. This shows 6.9.2(a). Condition 6.9.2(b) holds for Z' instead of X, since X is (μ', k)-valid, $n_0 \leq k$, and therefore $Z'^{\leq n_0} = X^{\leq n_0}$. This proves Claim 6.16. □

Claim 6.16 implies in particular that Z' is $(\mu, k+1)$-valid. This completes the proof of Lemma 6.14. □

One of the main consequences of Lemma 6.14 is that (μ, k)-valid oracles can be extended to (μ, k')-valid oracles for larger k'. We needed to include conditions 1(b) and 2(b) in Definition 6.9 in order to obtain this property. Otherwise it is possible that a certain way of extending the finite oracle X to some oracle X' has no extension to an infinite oracle O_2 so that $L(NM_i^{O_2}) \cap L(NM_j^{O_2}) = \emptyset$. If this happens, then by statement 6.9.1(a), for all extensions to an infinite oracle O_2, $A(O_2)$ and $B(O_2)$ would not be disjoint.

LEMMA 6.17. *If X is (μ, k)-valid, then for every $m > k$ there exists a (μ, m)-valid $Z \supseteq_k X$ such that $Z^{>k}$ contains only μ-codewords.*

Proof. It suffices to show the lemma for $m = k+1$. Let $Y = \emptyset$ and $N = \{0^{k+1}\}$. By Proposition 6.13.3, (Y, N) is a (μ, k)-reservation for X. Since $\|N\| = 1 \leq 2^{k/2}$ we can apply Lemma 6.14, and we obtain a $(\mu, k+1)$-valid $Z \supseteq_k X$ such that $Z^{>k}$ contains only μ-codewords. □

For a finite $X \subseteq \Sigma^*$, let $\ell(X) \stackrel{df}{=} \sum_{w \in X} |w|$.

LEMMA 6.18. *Let X be (μ, k)-valid and let $Z \supseteq_k X$ be (μ, m)-valid such that $m \geq k$ and $Z^{>k}$ contains only words of length $\equiv 0 \pmod 4$. For every $Y \subseteq Z$ and every $N \subseteq \overline{Z}$ there exists a (μ, k)-reservation (Y', N') for X such that $Y \subseteq Y'$, $N \subseteq N'$, $\ell(Y' \cup N') \leq 2 \cdot \ell(Y \cup N)$, $Y' \subseteq Z$, and $N' \subseteq \overline{Z}$.*

Proof. For every $Y \subseteq Z$ let

$$\mathcal{D}(Y) \stackrel{df}{=} \{q \mid Y^{>k} \text{ contains a } \mu\text{-codeword for } (i,t,x) \text{ and } q \in P^{\text{all}}_{i,t,x}\},$$

where $P_{i,t,x}$ is the lexicographically smallest path among all paths of $NM_i^Z(x)$ that are accepting and that are of length $\leq t$. Note that $\mathcal{D}(Y)$ is well-defined: If $Y^{>k} \subseteq Z$ contains a μ-codeword, then this has the form $00^{n_0}10^t1xy$ (resp., $10^{n_0}10^t1xy$), and there exist $i_0, j_0 \geq 1$ such that $(n_0, i_0, j_0) \in \mu$. Let $\mu_0 \stackrel{df}{=} \{(n', i', j') \in \mu \mid n' < n_0\}$. By statement 2 of Proposition 6.10, Z is $(\mu_0 \cup \{(n_0, i_0, j_0)\}, m)$-valid. From statement 6.9.1(a) it follows that the path $P_{i_0,t,x}$ (resp., $P_{j_0,t,x}$) exists.

If w is a μ-codeword for (i,t,x), then $|P_{i,t,x}| \leq t < |w|/4$. Therefore, when looking at the definition of $\mathcal{D}(Y)$, we see that the sum of lengths of q's that are induced by some μ-codeword w is at most $|w|/4$ (remember that we use nondeterministic machines that ask all queries in parallel). This shows the following.

CLAIM 6.19. *For all $Y \subseteq Z$, $\ell(\mathcal{D}(Y)) \leq \ell(Y)/4$, and words in $\mathcal{D}(Y)$ are not longer than the longest word in Y.*

Given Y and N, the procedure below computes the (μ, k)-reservation (Y', N').

```
1    Y_0 := Y
2    N_0 := N
3    c := 0
```

```
4    do
5        c := c + 1
6        Y_c := D(Y_{c-1}) ∩ Z
7        N_c := D(Y_{c-1}) ∩ Z̄
8    repeat until Y_c = N_c = ∅
9    Y' := Y_0 ∪ Y_1 ∪ ··· ∪ Y_c
10   N' := N_0 ∪ N_1 ∪ ··· ∪ N_c
```

Note that since all Y_c are subsets of Z, the expressions $\mathcal{D}(Y_{c-1})$ in lines 6 and 7 are defined. It is immediately clear that $Y \subseteq Y' \subseteq Z$ and $N \subseteq N' \subseteq \overline{Z}$. Therefore $Y' \cap N' = \emptyset$. From Claim 6.19 we obtain $\ell(Y_i \cup N_i) = \ell(\mathcal{D}(Y_{i-1})) \leq \ell(Y_{i-1})/4$ for $1 \leq i \leq c$. Therefore, the procedure terminates and $\ell(Y' \cup N') \leq 2 \cdot \ell(Y \cup N)$. It remains to show the following.

CLAIM 6.20. *(Y', N') is a (μ, k)-reservation for X.*

Clearly, $Y'^{\leq k} \subseteq X$ and $N'^{\leq k} \subseteq \overline{X}$. Moreover, $A(Y') \cap B(Y') = \emptyset$, since otherwise $A(Z) \cap B(Z) \neq \emptyset$, which is not possible, since Z is (μ, m)-valid. All words in $Y'^{>k}$ are of length $\equiv 0 \pmod 4$, since $Y' \subseteq Z$. Let $v \in Y'^{>k}$ be a μ-codeword for (i, t, x). More precisely, $v \in Y_{i'} \subseteq Z$ for a suitable $i' < c$. Z is (μ, m)-valid and v is a μ-codeword that belongs to Z. Therefore, as seen at the beginning of this proof, it follows that $NM_i^Z(x)$ accepts within t steps. Thus the path $P_{i,t,x}$ exists and we obtain $P^{\text{all}}_{i,t,x} \subseteq \mathcal{D}(Y_{i'})$. It follows that $P^{\text{yes}}_{i,t,x} \subseteq Y_{i'+1} \subseteq Y'$ and $P^{\text{no}}_{i,t,x} \subseteq N_{i'+1} \subseteq N'$. Therefore, $NM_i(x)$ has a positive path P with $|P| \leq t$, $P^{\text{yes}} \subseteq Y'$, and $P^{\text{no}} \subseteq N'$. This proves Claim 6.20 and finishes the proof of Lemma 6.18. □

For any (μ, k)-valid oracle either we can find a finite extension that makes the languages accepted by NM_i and NM_j not disjoint, or we can force these languages to be disjoint for all valid extensions.

PROPOSITION 6.21 (property P1). *Let $i, j \geq 1$ and let X be (μ, k)-valid. There exists an $l > k$ and a (μ', l)-valid $Y \supseteq_k X$, $\mu \preceq \mu'$ such that*

- *either for all $Z \supseteq_l Y$, $L(NM_i^Z) \cap L(NM_j^Z) \cap \Sigma^{\leq l} \neq \emptyset$, or*
- *$(i, j) \in \text{range}(\mu')$.*

This proposition tells us that if the first property does not hold, then by Definition 6.9, since Y is (μ', l)-valid, $L(NM_i^Z) \cap L(NM_j^Z) \cap \Sigma^{\leq m} = \emptyset$ for all (μ', m)-valid extensions Z of Y, where $m \geq l$.

Proof. By Lemma 6.17, we can assume that k is large enough so that $2 \cdot k^{i+j} < 2^{k/2}$. If $(i, j) \in \text{range}(\mu)$, then by Lemma 6.17, for $\mu' = \mu$ and $l = k + 1$ there exists a (μ', l)-valid $Y \supseteq_k X$. Otherwise we distinguish two cases.

Case 1. There exists an $l' > k$ and a (μ, l')-valid $Y' \supseteq_k X$ such that $L(NM_i^{Y'}) \cap L(NM_j^{Y'}) \cap \Sigma^{\leq l'} \neq \emptyset$. Choose some $x \in L(NM_i^{Y'}) \cap L(NM_j^{Y'}) \cap \Sigma^{\leq l'}$ and let P_i, P_j be accepting paths of the computations $NM_i^{Y'}(x)$, $NM_j^{Y'}(x)$, respectively. Note that $(P_i^{\text{yes}} \cup P_j^{\text{yes}}) \cap \Sigma^{>l'} = \emptyset$ and let $N \stackrel{df}{=} (P_i^{\text{no}} \cup P_j^{\text{no}}) \cap \Sigma^{>l'}$. By Proposition 6.13.3, $(\emptyset, N)$ is a (μ, l')-reservation for Y'. Since $\|N\| \leq 2 \cdot |x|^{i+j} \leq 2 \cdot l'^{i+j} < 2^{l'/2}$ we can apply Lemma 6.14. We obtain some $l \geq l' > k$ and some (μ, l)-valid $Y \supseteq_{l'} Y' \supseteq_k X$ such that $N \subseteq \Sigma^{\leq l}$ and $N \subseteq \overline{Y}$. Therefore, for every $Z \supseteq_l Y$ the computations $NM_i^Z(x)$ and $NM_j^Z(x)$ will accept at the paths P_i and P_j, respectively. Hence $L(NM_i^Z) \cap L(NM_j^Z) \cap \Sigma^{\leq l} \neq \emptyset$ for every $Z \supseteq_l Y$.

Case 2. For every $l' > k$ and every (μ, l')-valid $Y' \supseteq_k X$ it holds that $L(NM_i^{Y'}) \cap L(NM_j^{Y'}) \cap \Sigma^{\leq l'} = \emptyset$. By Lemma 6.17, there exists a (μ, l)-valid $Y \supseteq_k X$ where $l \stackrel{df}{=} k + 1$. Let $n_0 \stackrel{df}{=} l$, $i_0 \stackrel{df}{=} i$, $j_0 \stackrel{df}{=} j$, $\mu_0 \stackrel{df}{=} \mu$, and $\mu' \stackrel{df}{=} \mu_0 \cup \{(n_0, i_0, j_0)\}$. Observe that $n_0 > k \geq \mu_{\max}$, and therefore $\mu \preceq \mu'$. We show that Y is (μ', l)-valid.

We already know that $l \geq n_0$ and that Y is (μ_0, l)-valid. Since $i_0 > 0$ we only have to verify Definition 6.9.1. When looking at condition 6.9.1(a), we see that $4 \cdot |00^{n_0}10^t1x| \leq l$ is not possible, since $n_0 = l$. Therefore, condition 6.9.1(a) holds. Condition 6.9.1(b) follows from our assumption in Case 2. Therefore, Y is (μ', l)-valid. □

In order to show that $(C(O_2), D(O_2))$ is not symmetric we have to diagonalize against every possible reducing function, i.e., against every deterministic polynomial-time oracle transducer. The following proposition makes sure that this diagonalization is compatible with the notion of valid oracles.

PROPOSITION 6.22 (property P2). *Let $i \geq 1$ and let X be (μ, k)-valid. There exists an $l > k$ and a (μ, l)-valid $Y \supseteq_k X$ such that for all $Z \supseteq_l Y$, if $C(Z) \cap D(Z) = \emptyset$, then $(C(Z), D(Z))$ does not $\leq_m^{pp,O_2}$-reduce to $(D(Z), C(Z))$ via T_i^Z.*

Proof. By Lemma 6.17 we can assume that $k \equiv 0 \pmod 4$ and $(k+1)^i + 1 < 2^{(k+1)/2}$. Consider the computation $T_i^X(0^{k+1})$, let x be the output of this computation, and let N be the set of queries that are of length greater than k. If $|x| > k$, then additionally we add the word $0^{|x|}$ to N. Note that this yields an N such that $X \cap N = \emptyset$ and $\|N\| \leq (k+1)^i + 1 < 2^{(k+1)/2}$.

If $x \in C(X)$ (note that this implies $x = 0^{k'}$ for some $k' \leq k$), then choose some $y \in 0\Sigma^k - N$ and let $S \stackrel{df}{=} \{y\}$. In this case it holds that $0^{k+1} \in C(X \cup S) \wedge x \notin D(X \cup S)$. The right part of the conjunction holds, since X is (μ, k)-valid, and therefore $C(X) \cap D(X) = \emptyset$. Otherwise, if $x \notin C(X)$, then choose some $y \in 1\Sigma^k - N$ and let $S \stackrel{df}{=} \{y\}$. Here we obtain $0^{k+1} \in D(X \cup S) \wedge x \notin C(X \cup S)$. Together this means that we find some $y \in \Sigma^{k+1} - N$ such that with $S \stackrel{df}{=} \{y\}$ it holds that

$$(15) \quad [0^{k+1} \in C(X \cup S) \wedge x \notin D(X \cup S)] \ \vee \ [0^{k+1} \in D(X \cup S) \wedge x \notin C(X \cup S)].$$

Note that $S \subseteq \Sigma^{k+1}$ and $k+1 \not\equiv 0 \pmod 4$. Moreover, $C(S) \cap D(S) = \emptyset$ and for every n, $E_n(S) = \emptyset$, since by definition E_n depends only on oracle words of length $\equiv 2 \pmod 4$. From Proposition 6.10.4 it follows that $X \cup S$ is $(\mu, k+1)$-valid. So by Proposition 6.13.3, $(\emptyset, N)$ is a $(\mu, k+1)$-reservation for $X \cup S$. Since $\|N\| < 2^{(k+1)/2}$ we can apply Lemma 6.14. For $l \stackrel{df}{=} \max(\{|w| \mid w \in N\} \cup \{k+1\})$ we obtain a (μ, l)-valid $Y \supseteq_{k+1} X \cup S$ such that $N \subseteq \overline{Y}$ and $Y^{>k+1}$ contains only words of length $\equiv 0 \pmod 4$. Therefore, $T_i^Y(0^{k+1})$ computes x. Since all queries asked at this computation are of length $\leq l$, we obtain that $T_i^Z(0^{k+1})$ computes x for every $Z \supseteq_l Y$. Since $Y^{>k+1}$ does not contain words of length $\equiv 1 \pmod 4$ we have $C(Z) \cap \Sigma^{\leq l} = C(X \cup S)$ and $D(Z) \cap \Sigma^{\leq l} = D(X \cup S)$ for each $Z \supseteq_l Y$. Note that $k+1 \leq l$ and $|x| \leq l$. Therefore, by equation (15), the following holds for every $Z \supseteq_l Y$:

$$(16) \quad [0^{k+1} \in C(Z) \wedge T_i^Z(0^{k+1}) \notin D(Z)] \ \vee \ [0^{k+1} \in D(Z) \wedge T_i^Z(0^{k+1}) \notin C(Z)].$$

Hence, for every $Z \supseteq_l Y$, if $C(Z) \cap D(Z) = \emptyset$, then $(C(Z), D(Z))$ does not $\leq_m^{pp,O_2}$-reduce to $(D(Z), C(Z))$ via T_i^Z. □

For any (μ, k)-valid oracle, either we can find a finite extension that destroys NM_j's promise to be sparse, or we can force NM_j to be sparse for all valid extensions.

PROPOSITION 6.23 (property P3(a)). *Let $j \geq 1$ and let X be (μ, k)-valid. There exists an $l > k$ and a (μ', l)-valid $Y \supseteq_k X$, $\mu \preceq \mu'$, such that*

- *either for all $Z \supseteq_l Y$, $L(NM_j^Z) \notin \mathrm{SPARSE}_j$, or*
- *$(0, j) \in \mathrm{range}(\mu')$.*

This proposition tells us that if the first property does not hold, then there exists some n such that $(n, 0, j) \in \mu'$. In this case, from Definition 6.9 we obtain that for all

(μ', m)-valid extensions Z of Y it holds that $L(NM_j^Z) \cap \Sigma^{\leq m} \in \mathrm{SPARSE}_j$ and $E_n(Z)$ contains at most 2 words of every length.

Proof. By Lemma 6.17, we can assume that k is large enough so that $(k^j + j + 1) \cdot k^j < 2^{k/2}$. If $(0, j) \in \mathrm{range}(\mu)$, then by Lemma 6.17, for $\mu' = \mu$ and $l = k + 1$ there exists a (μ', l)-valid $Y \supseteq_k X$. Otherwise we distinguish two cases.

Case 1. There exists an $l' > k$ and a (μ, l')-valid $Y' \supseteq_k X$ such that $L(NM_j^{Y'}) \cap \Sigma^{\leq l'} \notin \mathrm{SPARSE}_j$. More precisely, there exists an $m \leq l'$ such that $\|L(NM_j^{Y'}) \cap \Sigma^m\| > m^j + j$. We choose $m^j + j + 1$ different words $x_0, \ldots, x_{m^j+j}$ from $L(NM_j^{Y'}) \cap \Sigma^m$. For $0 \leq i \leq m^j + j$, let P_i be an accepting path of the computation $NM_j^{Y'}(x_i)$. For all i, note that $P_i^{\mathrm{yes}} \cap \Sigma^{>l'} = \emptyset$ and let N be the union of all $P_i^{\mathrm{no}} \cap \Sigma^{>l'}$. By Proposition 6.13.3, $(\emptyset, N)$ is a (μ, l')-reservation for Y'. Since $\|N\| \leq (m^j+j+1) \cdot m^j \leq (l'^j + j + 1) \cdot l'^j < 2^{l'/2}$ we can apply Lemma 6.14. We obtain some $l \geq l' > k$ and some (μ, l)-valid $Y \supseteq_{l'} Y' \supseteq_k X$ such that $N \subseteq \Sigma^{\leq l}$ and $N \subseteq \overline{Y}$. Therefore, for every $Z \supseteq_l Y$ and every i, the computation $NM_j^Z(x_i)$ will accept at path P_i. Hence, for every $Z \supseteq_l Y$, $L(NM_j^Z) \notin \mathrm{SPARSE}_j$.

Case 2. For every $l' > k$ and every (μ, l')-valid $Y' \supseteq_k X$, it holds that $L(NM_j^{Y'}) \cap \Sigma^{\leq l'} \in \mathrm{SPARSE}_j$. By Lemma 6.17, there exists a (μ, l)-valid $Y \supseteq_k X$ with $l \stackrel{df}{=} k + 1$. Let $n_0 \stackrel{df}{=} l$, $i_0 \stackrel{df}{=} 0$, $j_0 \stackrel{df}{=} j$, $\mu_0 \stackrel{df}{=} \mu$, and $\mu' \stackrel{df}{=} \mu_0 \cup \{(n_0, i_0, j_0)\}$. Observe that $n_0 > k \geq \mu_{\max}$, and therefore $\mu_0 \preceq \mu'$. We will show that Y is (μ', l)-valid.

Since $l = \mu'_{\max}$ we have $l \geq \mu'_{\max}$. We already know $l \geq n_0$ and that Y is (μ_0, l)-valid. Since $i_0 = 0$, we only have to verify Definition 6.9.2. Since $l = n_0$ and $Y \subseteq \Sigma^{\leq l}$, we have $E_{n_0}(Y) = \emptyset$, which shows 6.9.2(a). Condition 6.9.2(b) follows from our assumption in Case 2. Therefore, Y is (μ', l)-valid. □

If NM_j is forced to be sparse for all valid extensions (Proposition 6.23), then we have to make sure that $L(NM_j)$ is not many-one-complete for NP ∩ SPARSE. We show that a certain E_n is sparse but is not many-one reducible to $L(NM_j)$. For this we have to diagonalize against every possible reducing function, i.e., against every deterministic polynomial-time oracle transducer. Proposition 6.25 makes sure that this diagonalization is possible. Before we give this proposition, we prove the following argument, which is used in the proofs for Proposition 6.25 and Lemma 6.29.

PROPOSITION 6.24. *Let X be (μ, k)-valid. Let (Y_1, N_1) be a $(\mu, k+1)$-reservation of some $(\mu, k+1)$-valid $Z_1 \supseteq_k X$, and let (Y_2, N_2) be a $(\mu, k+1)$-reservation of some $(\mu, k+1)$-valid $Z_2 \supseteq_k X$ such that ${Y_1}^{>k+1} \cup {Y_2}^{>k+1}$ contains only μ-codewords. If $\|N_1 \cup N_2\| \leq 2^{(k+1)/2}$, $Y_1 \cap N_2 = Y_2 \cap N_1 = \emptyset$, and $X' \stackrel{df}{=} X \cup {Y_1}^{=k+1} \cup {Y_2}^{=k+1}$ is $(\mu, k+1)$-valid, then $A(Y_1 \cup Y_2) \cap B(Y_1 \cup Y_2) = \emptyset$.*

Proof. In order to see that (Y_1, N_1) is a $(\mu, k+1)$-reservation for X', it suffices to show that ${Y_1}^{=k+1} \subseteq X'$ and ${N_1}^{=k+1} \subseteq \overline{X'}$. The first inclusion holds by the definition of X'. The second one holds, since otherwise either $Y_1 \cap N_1 \neq \emptyset$ (not possible since (Y_1, N_1) is a $(\mu, k+1)$-reservation) or $Y_2 \cap N_1 \neq \emptyset$ (not possible by assumption). It follows that (Y_1, N_1) is a $(\mu, k+1)$-reservation for X', and, analogously, (Y_2, N_2) is a $(\mu, k+1)$-reservation for X'.

Assume that $A(Y_1 \cup Y_2) \cap B(Y_1 \cup Y_2) \neq \emptyset$. Choose a shortest $w \in A(Y_1 \cup Y_2) \cap B(Y_1 \cup Y_2)$. Hence, there exist $y_0, y_1 \in \Sigma^{3|w|+3}$ such that $0wy_0, 1wy_1 \in Y_1 \cup Y_2$. Let $m \stackrel{df}{=} |0wy_0| - 1$. We show $m \geq k + 1$. Otherwise, if $m \leq k$, then $|0wy_0| = |1wy_1| \leq k + 1$. It follows that $0wy_0, 1wy_1 \in X'$, since (Y_1, N_1) and (Y_2, N_2) are $(\mu, k+1)$-reservations for X'. This implies $w \in A(X') \cap B(X')$, which is not possible. Therefore, $m \geq k + 1$.

By Proposition 6.13.5, $(Y_1^{\leq m}, N_1^{\leq m})$ and $(Y_2^{\leq m}, N_2^{\leq m})$ are $(\mu, k+1)$-reservations for X'. Let $Y \stackrel{df}{=} Y_1^{\leq m} \cup Y_2^{\leq m}$ and $N \stackrel{df}{=} N_1^{\leq m} \cup N_2^{\leq m}$. We show that (Y, N) is a $(\mu, k+1)$-reservation for X'. For this it suffices to verify $Y \cap N = \emptyset$ and $A(Y) \cap B(Y) = \emptyset$. The first equality holds, since otherwise either $Y_1 \cap N_2 \neq \emptyset$ or $Y_2 \cap N_1 \neq \emptyset$, which is not possible by assumption. If $A(Y) \cap B(Y) \neq \emptyset$, then there exists some $w' \in A(Y) \cap B(Y)$ such that $|w'| < |w|$. This is not possible, since $A(Y) \cap B(Y) \subseteq A(Y_1 \cup Y_2) \cap B(Y_1 \cup Y_2)$ and since w was chosen as short as possible. Therefore, (Y, N) is a $(\mu, k+1)$-reservation for X'.

Note that $\|N\| \leq 2^{(k+1)/2}$. By Lemmas 6.14 and 6.17, there exists a (μ, m)-valid $Z \supseteq_{k+1} X'$ such that $Y \subseteq Z$ and $N \subseteq \overline{Z}$. We know that $|0wy_0| > k+1$ and $0wy_0 \in Y_1 \cup Y_2$. Without loss of generality we assume $0wy_0 \in Y_1$. So by assumption, $0wy_0$ is a μ-codeword. Hence, $w = 0^n 10^t 1x$ for suitable n, t, x such that n is in the domain of μ. Let $\mu(n) = (i, j)$, where $i, j \geq 1$. From $0wy_0 \in Y_1$ it follows that $NM_i(x)$ has a positive path P such that $|P| \leq t$, $P^{\text{yes}} \subseteq Y_1$, and $P^{\text{no}} \subseteq N_1$. Since elements from P^{yes} and P^{no} are of length $\leq t \leq m$, we obtain $P^{\text{yes}} \subseteq Y \subseteq Z$, and $P^{\text{no}} \subseteq N \subseteq \overline{Z}$. It follows that $NM_i^Z(x)$ accepts. Analogously (i.e., with the help of $1wy_1$) we obtain that $NM_j^Z(x)$ accepts. This shows $x \in L(NM_i^Z) \cap L(NM_j^Z) \cap \Sigma^{\leq m}$, which contradicts Proposition 6.10.5. □

PROPOSITION 6.25 (property P3(b)). *Let $i, j \geq 1$ and let X be (μ, k)-valid such that for a suitable n, $\mu(n) = (0, j)$. There exists an $l > k$ and a (μ, l)-valid $Y \supseteq_k X$ such that for all $Z \supseteq_l Y$, $E_n(Z)$ does not $\leq_m^{p,Z}$-reduce to $L(NM_j^Z)$ via f_i^Z.*

Proof. Let $\alpha \stackrel{df}{=} (k+1)^i$, $\beta \stackrel{df}{=} (\alpha+1) \cdot (\alpha^j + j) + 1$, and $\gamma \stackrel{df}{=} \beta \cdot (2 \cdot \alpha^j + 2)$. Note that if i and j are considered as constants, then the values of α, β, and γ are polynomial in $k+1$. By Lemma 6.17, we can assume that $k \equiv 1 \pmod 4$, and that k is large enough such that $n + 2 + \log \gamma \leq (k+1)/2$ and $(2 \cdot \alpha^j + 2) \cdot \gamma < 2^{(k+1)/2}$.

Let $x_1, \ldots, x_\gamma$ be the binary representations (possibly with leading zeros) of $1, \ldots, \gamma$, respectively, such that for all r, $|0^n 1 x_r| = (k+1)/2$. For $1 \leq r \leq \gamma$, let $z_r \stackrel{df}{=} f_i^X(0^n 1 x_r)$ and note that the lengths of these words are bounded by α. We consider two cases.

Case 1. There exist a, b such that $1 \leq a < b \leq \gamma$ and $z_a = z_b$. Let N be the set of queries of length $> k$ that are asked during the computations $f_i^X(0^n 1 x_a)$ and $f_i^X(0^n 1 x_b)$. Note that these are negative queries. Observe that $\|N\| \leq 2 \cdot \alpha < 2^{(k+1)/2}$ and choose a word y_a of length $(k+1)/2$ such that $0^n 1 x_a y_a \notin N$. Let $S \stackrel{df}{=} \{0^n 1 x_a y_a\}$. It follows that $C(S) \cap D(S) = \emptyset$. Moreover, for all $n' \geq 1$, $\|E_{n'}(S)\| \leq 1$. From Proposition 6.10.4 it follows that $X' \stackrel{df}{=} X \cup S$ is $(\mu, k+1)$-valid. By Proposition 6.13.3, $(\emptyset, N)$ is a $(\mu, k+1)$-reservation for X'. By Lemma 6.14, there exists a (μ, l)-valid $Y \supseteq_{k+1} X'$ such that $N \subseteq \Sigma^{\leq l}$ and $N \subseteq \overline{Y}$. Therefore, for all $Z \supseteq_l Y$ it holds that $f_i^Z(0^n 1 x_a) = f_i^Z(0^n 1 x_b) = z_a$. Moreover, $0^n 1 x_a \in E_n(Z)$ and $0^n 1 x_b \notin E_n(Z)$. This shows that for all $Z \supseteq_l Y$, $E_n(Z)$ does not $\leq_m^{p,Z}$-reduce to $L(NM_j^Z)$ via f_i^Z.

Case 2. For $1 \leq r \leq \gamma$, all z_r are pairwise different. The remaining part of the proof deals with this case. Until the end of the proof, r will always be such that $1 \leq r \leq \gamma$. For every r, define the following set:

$$L_r \stackrel{df}{=} \{(Y_r, N_r) \mid (Y_r, N_r) \text{ is a } (\mu, k+1)\text{-reservation for some } (\mu, k+1)\text{-valid } Z \supseteq_k X \text{ such that } Z^{=k+1} \subseteq 0^n 1 \Sigma^*,\ \|Z^{=k+1}\| \leq 1,\ Y_r{}^{>k+1} \text{ contains only } \mu\text{-codewords},\ \ell(Y_r \cup N_r) \leq 2 \cdot \alpha^j,\ \text{and } NM_j(z_r) \text{ has a positive path } P_r \text{ such that } P_r^{\text{yes}} \subseteq Y_r \text{ and } P_r^{\text{no}} \subseteq N_r\}.$$

In the following we consider vectors $v = ((Y_{r_1}, N_{r_1}), (Y_{r_2}, N_{r_2}), \ldots, (Y_{r_s}, N_{r_s}))$ such that $1 \leq s \leq \beta$, all r_a are from $[1, \gamma]$ and are pairwise different, and $(Y_{r_a}, N_{r_a}) \in$

L_{r_a}. Such vectors v are called *vectors of reservations from* $L_1, \ldots, L_\gamma$. We say that v *has a conflict* if there exist a, b such that $1 \le a < b \le s$, and either $Y_{r_a} \cap N_{r_b} \neq \emptyset$ or $N_{r_a} \cap Y_{r_b} \neq \emptyset$. In this case we also say that the reservations (Y_{r_a}, N_{r_a}) and (Y_{r_b}, N_{r_b}) *conflict*. Now we are going to prove three claims. After this, with Claim 6.28 at hand, we are able to finish Case 2.

CLAIM 6.26. *Let* $(Y_a, N_a) \in L_a$ *and* $(Y_b, N_b) \in L_b$. *If* (Y_a, N_a) *and* (Y_b, N_b) *do not conflict, then* $A(Y_a \cup Y_b) \cap B(Y_a \cup Y_b) = \emptyset$.

Assume that (Y_a, N_a) and (Y_b, N_b) do not conflict. Let $S \stackrel{df}{=} {Y_a}^{=k+1} \cup {Y_b}^{=k+1}$ and $X' \stackrel{df}{=} X \cup S$. From the definition of L_a and L_b it follows that $\|S\| \le 2$. Therefore, for all $n' \ge 1$, $\|E_{n'}(S)\| \le 2$. Moreover, $C(S) = D(S) = \emptyset$, since C and D depend only on oracle words of length $\equiv 1 \pmod 4$. From Proposition 6.10.4, we obtain that X' is $(\mu, k{+}1)$-valid. Note that $\|N_a \cup N_b\| \le 2^{(k+1)/2}$, since $\|N_a \cup N_b\| \le \ell(N_a) + \ell(N_b) + 2 \le 2(2\alpha^j + 1) \le \gamma(2\alpha^j + 1)$. By assumption, $Y_a \cap N_b = Y_b \cap N_a = \emptyset$. Therefore, from Proposition 6.24 it follows that $A(Y_a \cup Y_b) \cap B(Y_a \cup Y_b) = \emptyset$. This shows Claim 6.26.

CLAIM 6.27. *Every* β*-dimensional vector of reservations has a conflict.*

Proof. Assume that there exists a vector of reservations

$$v = ((Y_{r_1}, N_{r_1}), (Y_{r_2}, N_{r_2}), \ldots, (Y_{r_\beta}, N_{r_\beta}))$$

such that v has no conflict. Let $\mu' \stackrel{df}{=} \{(n', i', j') \in \mu \mid n' < n\}$. Note that X is (μ', k)-valid and also $(\mu' \cup \{n, 0, j\}, k)$-valid (Proposition 6.10.2). Let $Y \stackrel{df}{=} \bigcup_{1 \le a \le \beta} Y_{r_a}$, $N \stackrel{df}{=} \bigcup_{1 \le a \le \beta} N_{r_a}$, and $X' \stackrel{df}{=} X \cup Y^{=k+1}$. We show that X' is $(\mu', k+1)$-valid. Since C and D depend only on oracle words of length $\equiv 1 \pmod 4$, we have $C(Y^{=k+1}) = D(Y^{=k+1}) = \emptyset$. Moreover, since n is not in the domain of μ' and since all words in $Y^{=k+1}$ have the prefix $0^n 1$, for all $(n', 0, j') \in \mu'$ it holds that $E_{n'}(Y^{=k+1}) = \emptyset$. Therefore, from Proposition 6.10.4 it follows that X' is $(\mu', k+1)$-valid.

Let us show that for $1 \le a \le \beta$, (Y_{r_a}, N_{r_a}) is a $(\mu', k+1)$-reservation for X'. By definition, (Y_{r_a}, N_{r_a}) is a $(\mu, k+1)$-reservation for some $(\mu, k+1)$-valid $Z \supseteq_k X$. Since every μ'-codeword is a μ-codeword, it suffices to verify ${Y_{r_a}}^{=k+1} \subseteq X'$ and ${N_{r_a}}^{=k+1} \subseteq \overline{X'}$. The first inclusion holds by the definition of X'. If the latter inclusion does not hold, then ${N_{r_a}}^{=k+1} \cap Y^{=k+1} \neq \emptyset$. Since $N_{r_a} \cap Y_{r_a} = \emptyset$, it follows that $N_{r_a} \cap Y_{r_b} \neq \emptyset$ for some $b \neq a$. This implies that v has a conflict, which is not possible by our assumption. This shows that for all a, if $1 \le a \le \beta$, then (Y_{r_a}, N_{r_a}) is a $(\mu', k+1)$-reservation for X'.

We show that (Y, N) is a $(\mu', k{+}1)$-reservation for X'. All (Y_{r_a}, N_{r_a}) are $(\mu', k{+}1)$-reservations that do not conflict with each other. From this we immediately obtain that $Y \cap N = \emptyset$, $Y^{\le k+1} \subseteq X'$, $N^{\le k+1} \subseteq \overline{X'}$, and all words in $Y^{>k+1}$ are of length $\equiv 0 \pmod 4$. If $A(Y) \cap B(Y) \neq \emptyset$, then there exist a, b such that $A(Y_{r_a} \cup Y_{r_b}) \cap B(Y_{r_a} \cup Y_{r_b}) \neq \emptyset$. This contradicts Claim 6.26. Therefore, $A(Y) \cap B(Y) = \emptyset$. Finally, if $w \in Y^{>k+1}$ is a μ'-codeword for (i', t', x'), then there exists some a such that $w \in {Y_{r_a}}^{>k+1}$. Since (Y_{r_a}, N_{r_a}) is a $(\mu', k+1)$-reservation, $NM_{i'}(x')$ has a positive path P such that $|P| \le t'$, $P^{\text{yes}} \subseteq Y_{r_a} \subseteq Y$, and $P^{\text{no}} \subseteq N_{r_a} \subseteq N$. This shows that (Y, N) is a $(\mu', k+1)$-reservation for X'.

By definition, for all r and all $(Y_r, N_r) \in L_r$ it holds that $\ell(Y_r \cup N_r) \le 2 \cdot \alpha^j$. Therefore, $\|N_r\| \le 2 \cdot \alpha^j + 1$ and it follows that $\|N\| \le \beta \cdot (2 \cdot \alpha^j + 1) \le 2^{(k+1)/2}$. By Lemmas 6.14 and 6.17 there exists some (μ', m)-valid $Z \supseteq_{k+1} X'$ such that $Y \cup N \subseteq \Sigma^{\le m}$, $Y \subseteq Z$, $N \subseteq \overline{Z}$, and $m \ge \alpha$. From the definition of the sets L_r it follows that for all a, if $1 \le a \le \beta$, then $NM_j^Z(z_{r_a})$ accepts. The length of all z_{r_a} is bounded by α. So there exists a length l such that $0 \le l \le \alpha$ and at least $\beta/(\alpha + 1) > (\alpha^j + j) \ge l^j + j$

of the words z_{r_a} are of length l. Hence $\|L(NM_j^Z) \cap \Sigma^l\| > l^j + j$, and therefore $L(NM_j^Z) \cap \Sigma^{\leq m} \notin \text{SPARSE}_j$.

We know that X is $(\mu' \cup \{n, 0, j\}, k)$-valid. Moreover, $m \geq k \geq n$ and Z is (μ', m)-valid such that $Z^{\leq k} = X^{\leq k}$, and therefore $Z^{\leq n} = X^{\leq n}$. From Definition 6.9.2(b) it follows that $L(NM_j^Z) \cap \Sigma^{\leq m} \in \text{SPARSE}_j$. This contradicts our observation in the last paragraph and finishes the proof of Claim 6.27. □

CLAIM 6.28. *There exist some r and an $N \subseteq \Sigma^{>k}$ such that $\|N\| \leq (2 \cdot \alpha^j + 2) \cdot \gamma$ and, for every (μ, m)-valid $Z \supseteq_k X$, if $m > k$, $N \subseteq \overline{Z} \cap \Sigma^{\leq m}$, $Z^{=k+1} \subseteq 0^n 1 \Sigma^*$, $\|Z \cap \Sigma^{k+1}\| \leq 1$, and $Z^{>k+1}$ contains only μ-codewords, then $NM_j^Z(z_r)$ rejects.*

Proof. We use the following algorithm to create the set N. Note that this algorithm modifies the sets L_r. This will decrease the number of possible vectors of reservations from $L_1, \ldots, L_\gamma$.

```
1  N(0) := ∅, R(0) := ∅, i := 0
2  while (all L_r ≠ ∅)
3     i := i + 1
4     choose the largest d such that there exists a
         d-dimensional vector v = ((Y_{r_1}, N_{r_1}), ..., (Y_{r_d}, N_{r_d})) of
         reservations from L_1, ..., L_γ such that
         v has no conflict
5     R(i) := R(i − 1) ∪ {r_1, r_2, ..., r_d}
6     N(i) := N(i − 1) ∪ Y_{r_1}^{>k} ∪ N_{r_1}^{>k} ∪ ··· ∪ Y_{r_d}^{>k} ∪ N_{r_d}^{>k}
7     for every r and every (Y_r, N_r) ∈ L_r:
         remove (Y_r, N_r) if Y_r ∩ N(i) ≠ ∅
8  end while
9  N := N(i)
```

Let $i \geq 1$ and consider the algorithm after the ith iteration of the *while* loop. We claim that for every $r \notin R(i)$ and every (Y_r, N_r) that remains in L_r, it holds that $N_r \cap (N(i) - N(i-1)) \neq \emptyset$. Otherwise, there exist r and (Y_r, N_r) such that $r \notin R(i)$, $(Y_r, N_r) \in L_r$, $N_r \cap (N(i) - N(i-1)) = \emptyset$, and (Y_r, N_r) has not been removed in step 7. Therefore, $Y_r \cap N(i) = \emptyset$, which implies $Y_r \cap (N(i) - N(i-1)) = \emptyset$. Together with our assumption, this gives us $(Y_r \cup N_r) \cap (N(i) - N(i-1)) = \emptyset$. By step 6 this means that (Y_r, N_r) does not conflict with any reservation in v. Therefore, with $((Y_r, N_r), (Y_{r_1}, N_{r_1}), \ldots, (Y_{r_d}, N_{r_d}))$ we found a $(d+1)$-dimensional vector of reservations that has no conflict. This contradicts the choice of v in step 4. Therefore, for every $r \notin R(i)$ and every (Y_r, N_r) that remains in L_r, it holds that $N_r \cap (N(i) - N(i-1)) \neq \emptyset$. It follows that after l iterations of the *while* loop, for every $r \notin R(l)$ and every (Y_r, N_r) that remains in L_r, it holds that $\|N_r\| \geq l$.

By Claim 6.27 and the choice of d in step 4 we have $d < \beta$. Therefore, after $(2 \cdot \alpha^j + 2)$ iterations, $\|R(i)\| < (2 \cdot \alpha^j + 2) \cdot \beta = \gamma$. So during the first $(2 \cdot \alpha^j + 2)$ iterations i there always exists an $r \notin R(i)$. Moreover, for every r and every $(Y_r, N_r) \in L_r$, it holds that $\ell(Y_r \cup N_r) \leq 2 \cdot \alpha^j$, and therefore $\|N_r\| \leq 2 \cdot \alpha^j + 1$. From the conclusion of the previous paragraph it follows that the *while* loop iterates at most $2 \cdot \alpha^j + 2$ times. This shows that the algorithm terminates. Since $d < \beta$, for all $i \geq 1$ it holds that $\|N(i) - N(i-1)\| < \beta \cdot (2 \cdot \alpha^j + 1) \leq \gamma$. Therefore, $\|N\| \leq (2 \cdot \alpha^j + 2) \cdot \gamma$ and $N \subseteq \Sigma^{>k}$ when the algorithm terminates.

So we have a set N of the required size and an r such that $L_r = \emptyset$. We show that N and r satisfy Claim 6.28. Assume that for some $m \geq k+1$ there exists a (μ, m)-valid $Z \supseteq_k X$ such that $N \subseteq \overline{Z} \cap \Sigma^{\leq m}$, $Z^{=k+1} \subseteq 0^n 1 \Sigma^*$, $\|Z \cap \Sigma^{k+1}\| \leq 1$, $Z^{>k+1}$ contains only μ-codewords, and $NM_j^Z(z_r)$ accepts. Let P_r be an accepting path of $NM_j^Z(z_r)$.

Let $Z' \stackrel{df}{=} Z^{\leq k+1}$. From Proposition 6.10.6 it follows that Z' is $(\mu, k+1)$-valid (since $k+1 > k \geq \mu_{\max}$). $Z^{>k+1}$ contains only words of length $\equiv 0 \pmod 4$, since it contains only μ-codewords. So we can apply Lemma 6.18 (for $X = Z'$, $Y = P_r^{\text{yes}}$, and $N = P_r^{\text{no}}$). We obtain a $(\mu, k+1)$-reservation (Y', N') for Z' such that $P_r^{\text{yes}} \subseteq Y'$, $P_r^{\text{no}} \subseteq N'$, $\ell(Y' \cup N') \leq 2 \cdot \ell(P_r^{\text{yes}} \cup P_r^{\text{no}}) \leq 2 \cdot \alpha^j$, $Y' \subseteq Z$, and $N' \subseteq \overline{Z}$. Together with $N \subseteq \overline{Z}$, this implies

$$Y' \cap N = \emptyset. \tag{17}$$

We show that at the beginning of the algorithm, (Y', N') must have been in L_r . Since $Z^{>k+1}$ contains only μ-codewords and since $Y' \subseteq Z$, then $Y'^{>k+1}$ also contains only μ-codewords. Moreover, $Z'^{=k+1} = Z^{=k+1} \subseteq 0^n 1 \Sigma^*$ and $\|Z' \cap \Sigma^{k+1}\| = \|Z \cap \Sigma^{k+1}\| \leq 1$. By our assumption, P_r is a positive path of $NM_j(z_r)$, and it holds that $P_r^{\text{yes}} \subseteq Y'$ and $P_r^{\text{no}} \subseteq N'$. It follows that (Y', N') must have been in L_r.

Since $L_r = \emptyset$ when the algorithm terminates, (Y', N') has been removed during some iteration i. This implies that during that iteration, $Y' \cap N(i) \neq \emptyset$ (by line 7). Moreover, by line 9, $N(i) \subseteq N$. This implies $Y' \cap N \neq \emptyset$, which contradicts (17). This proves Claim 6.28. □

Now we finish Case 2. Let r and N be as in Claim 6.28. Choose a word y_r of length $(k+1)/2$ such that $0^n 1 x_r y_r \notin N$. Let $S \stackrel{df}{=} \{0^n 1 x_r y_r\}$. It follows that $C(S) = D(S) = \emptyset$. Moreover, for all $n' \geq 1$, $\|E_{n'}(S)\| \leq 1$. From Proposition 6.10.4 it follows that $X' \stackrel{df}{=} X \cup S$ is $(\mu, k+1)$-valid. By Proposition 6.13.3, $(\emptyset, N)$ is a $(\mu, k+1)$-reservation for X'. Note that $\|N\| \leq (2 \cdot \alpha^j + 2) \cdot \gamma < 2^{(k+1)/2}$. Therefore, by Lemmas 6.14 and 6.17 there exists an $l \geq \alpha^j$ and a (μ, l)-valid $Y \supseteq_{k+1} X'$ such that $N \subseteq \overline{Y} \cap \Sigma^{\leq l}$ and $Y^{>k+1}$ contains only μ-codewords. From Claim 6.28 it follows that $NM_j^Y(z_r)$ rejects. The computation times of $f_i^Y(0^n 1 x_r)$ and $NM_j^Y(z_r)$ are bounded by $\alpha^j \leq l$. Therefore, for all $Z \supseteq_l Y$ it holds that $f_i^Z(0^n 1 x_r) = z_r$, $0^n 1 x_r \in E_n(Z)$, and $NM_j^Z(z_r)$ rejects. This shows that $E_n(Z)$ does not $\leq_m^{p,Z}$-reduce to $L(NM_j^Z)$ via f_i^Z. This finishes the proof of Proposition 6.25. □

Recall that we want to construct the oracle in a way such that $(A(O_2), B(O_2))$ is not $\leq_T^{pp,O_2}$-hard for NP^{O_2}. We have seen that it suffices to construct $F(O_2)$ such that it does not $\leq_T^{pp}$-reduce to $(A(O_2), B(O_2))$. We prevent $F(O_2) \leq_T^{pp} (A(O_2), B(O_2))$ via M_i as follows: We consider the computation $M_i(0^n)$, where the machine can ask queries to the pair $(A(X), B(X))$. In Lemma 6.29 we show that each query to this pair can be forced to be either in the complement of $A(X)$ or in the complement of $B(X)$. For this forcing it is enough to reserve polynomially many words for the complement of X. If we forced the query to be in the complement of $A(X)$, then the oracle can safely answer that the query belongs to $B(X)$. Otherwise it can safely answer that the query belongs to $A(X)$. After forcing all queries of the computation, we add an unreserved word to $F(X)$ if and only if the computation rejects. This will show that $F(X)$ does not $\leq_T^{pp}$-reduce to $(A(X), B(X))$ via M_i (Proposition 6.32).

LEMMA 6.29. *Let $k \equiv 2 \pmod 4$ and let X be (μ, k)-valid. For every $q \in \Sigma^*$, $|q| \leq 2^{k/2-4} - 2$, there exists an $N \subseteq \Sigma^{>k}$ such that $\|N\| \leq (8 \cdot |q| + 10)^2$ and one of the following properties holds:*

1. *For all (μ, m)-valid $Z \supseteq_k X$, if $m > k$, $N \subseteq \overline{Z}$, and $Z^{>k+1}$ contains only μ-codewords, then $q \notin A(Z)$.*
2. *For all (μ, m)-valid $Z \supseteq_k X$, if $m > k$, $N \subseteq \overline{Z}$, and $Z^{>k+1}$ contains only μ-codewords, then $q \notin B(Z)$.*

Proof. We can assume that $q = 0^n 10^t 1x$ for suitable n, t, x. Otherwise, q cannot belong to $A(Z) \cup B(Z)$ for all oracles Z, and we are done. Define the following sets:

$L_A \stackrel{df}{=} \{(Y_A, N_A) \mid (Y_A, N_A)$ is a $(\mu, k{+}1)$-reservation for some $(\mu, k{+}1)$-valid $Z \supseteq_k X$, ${Y_A}^{>k+1}$ contains only μ-codewords, $\ell(Y_A \cup N_A) \leq 8(|q|+1)$, and $(\exists y \in \Sigma^{3|q|+3})[0qy \in Y_A]\}$,

$L_B \stackrel{df}{=} \{(Y_B, N_B) \mid (Y_B, N_B)$ is a $(\mu, k{+}1)$-reservation for some $(\mu, k{+}1)$-valid $Z \supseteq_k X$, ${Y_B}^{>k+1}$ contains only μ-codewords, $\ell(Y_B \cup N_B) \leq 8(|q|+1)$, and $(\exists y \in \Sigma^{3|q|+3})[1qy \in Y_B]\}$.

We say that $(Y_A, N_A) \in L_A$ and $(Y_B, N_B) \in L_B$ *conflict* if and only if $Y_A \cap N_B \neq \emptyset$ or $N_A \cap Y_B \neq \emptyset$. Note that if (Y_A, N_A) and (Y_B, N_B) conflict, then even $Y_A \cap N_B \cap \Sigma^{>k} \neq \emptyset$ or $N_A \cap Y_B \cap \Sigma^{>k} \neq \emptyset$.

Claim 6.30. *Every $(Y_A, N_A) \in L_A$ conflicts with every $(Y_B, N_B) \in L_B$.*

Proof. Assume that there exist $(Y_A, N_A) \in L_A$ and $(Y_B, N_B) \in L_B$ that do not conflict. Let $Y' \stackrel{df}{=} Y_A \cup Y_B$, $N' \stackrel{df}{=} N_A \cup N_B$ and $S \stackrel{df}{=} {Y_A}^{=k+1} \cup {Y_B}^{=k+1}$.

We show that (Y', N') is a $(\mu, k+1)$-reservation for $X' \stackrel{df}{=} X \cup S$. Since $k \equiv 2 \pmod 4$ and $S \subseteq \Sigma^{k+1}$, it holds that $C(S) = D(S) = \emptyset$ and, for all $n' \geq 1$, $E_{n'}(S) = \emptyset$. From Proposition 6.10.4, it follows that X' is $(\mu, k+1)$-valid. Note that $\|N_A \cup N_B\| \leq 2^{(k+1)/2}$, since $\|N_A \cup N_B\| \leq \ell(N_A) + \ell(N_B) + 2 \leq 16|q| + 18 \leq 2^{k/2}$. By assumption, $Y_A \cap N_B = Y_B \cap N_A = \emptyset$. From Proposition 6.24 it follows that $A(Y_A \cup Y_B) \cap B(Y_A \cup Y_B) = \emptyset$. Therefore, it remains to verify $Y' \cap N' = \emptyset$, ${Y'}^{=k+1} \subseteq X'$, and ${N'}^{=k+1} \subseteq \overline{X'}$. The first condition holds, since (Y_A, N_A) and (Y_B, N_B) do not conflict. The second one holds by the definition of X'. Finally, ${N'}^{=k+1} \subseteq \overline{X'}$ holds, since otherwise ${N'}^{=k+1} \cap S \neq \emptyset$, and therefore $Y' \cap N' \neq \emptyset$. This shows that (Y', N') is a $(\mu, k+1)$-reservation for X'.

From the definition of L_A and L_B it follows that $\|N'\| \leq 16 \cdot |q| + 18 \leq 2^{k/2}$. By Lemma 6.14, there exist an $m \geq k+1$ and a (μ, m)-valid $Z \supseteq_{k+1} X'$ such that $Y' \subseteq Z$. Since $(Y_A, N_A) \in L_A$ and $(Y_B, N_B) \in L_B$, there exist $y_0, y_1 \in \Sigma^{3|q|+3}$ such that $0qy_0 \in Y_A \subseteq Y' \subseteq Z$ and $1qy_1 \in Y_B \subseteq Y' \subseteq Z$. Therefore, $q \in A(Z) \cap B(Z)$, which contradicts the fact that Z is (μ, m)-valid. This proves Claim 6.30. □

We use the following algorithm to create the set N as claimed in the statement of this lemma.

```
1    N := ∅
2    while (L_A ≠ ∅ and L_B ≠ ∅)
3        choose some (Y'_A, N'_A) ∈ L_A
4        N := N ∪ Y'_A^{>k} ∪ N'_A^{>k}
5        for every (Y_A, N_A) ∈ L_A
6            remove (Y_A, N_A) if Y_A ∩ (Y'_A^{>k} ∪ N'_A^{>k}) ≠ ∅
7        for every (Y_B, N_B) ∈ L_B
8            remove (Y_B, N_B) if Y_B ∩ (Y'_A^{>k} ∪ N'_A^{>k}) ≠ ∅
9    end while
```

We claim that after l iterations of the *while* loop, for every $(Y_B, N_B) \in L_B$, $\|N_B\| \geq l$. If this claim is true, the while loop iterates at most $8 \cdot |q| + 10$ times, since for any $(Y_B, N_B) \in L_B$, $\ell(N_B) \leq 8 \cdot |q| + 8$, and therefore $\|N_B\| \leq 8 \cdot |q| + 9$. On the other hand, during each iteration, N is increased by at most $8 \cdot |q| + 9$ strings. Therefore, $\|N\| \leq (8 \cdot |q| + 10)^2$ and $N \subseteq \Sigma^{>k}$ when this algorithm terminates.

Claim 6.31. *After l iterations of the* while *loop, for every (Y_B, N_B) that remains in L_B, $\|N_B\| \geq l$.*

Proof. For every l, let us denote the pair that is chosen during the lth iteration in step 3 by (Y_A^l, N_A^l). By Claim 6.30, every (Y_B, N_B) that belongs to L_B at the beginning of this iteration conflicts with (Y_A^l, N_A^l), i.e., $N_A^l \cap Y_B \cap \Sigma^{>k} \neq \emptyset$ or $Y_A^l \cap N_B \cap \Sigma^{>k} \neq \emptyset$. If $N_A^l \cap Y_B \cap \Sigma^{>k} \neq \emptyset$, then (Y_B, N_B) will be removed from L_B in step 8. Otherwise, $Y_A^l \cap N_B \cap \Sigma^{>k}$ is not empty, and therefore there exists a lexicographically smallest word w_l in this set. In this case, (Y_B, N_B) will not be removed from L_B; we say that (Y_B, N_B) *survives* the lth iteration *due to the word* w_l. Note that (Y_B, N_B) can survive only due to a word that belongs to N_B. We will use this fact to prove that $\|N_B\| \geq l$ after l iterations.

We show now that any pair (Y_B, N_B) that is left in L_B after l iterations survives each of these iterations due to a different word. Since these words all belong to N_B, this will complete the proof of the claim. Assume that there exist iterations l and l' with $l < l'$ such that $w_l = w_{l'}$. Then $w_l \in Y_A^l \cap N_B \cap \Sigma^{>k}$ and $w_{l'} \in Y_A^{l'} \cap N_B \cap \Sigma^{>k}$. Therefore, $Y_A^l \cap Y_A^{l'} \cap \Sigma^{>k} \neq \emptyset$. So the pair $(Y_A^{l'}, N_A^{l'})$ should have been removed in iteration l (step 6) and cannot be chosen at the beginning of iteration l', as claimed. Hence, $w_l \neq w_{l'}$. This proves Claim 6.31. □

Therefore, we now have a set N of the required size such that either L_A or L_B will be empty. Assume that L_A is empty; we will show that Lemma 6.29.1 holds. Analogously we show that if L_B is empty, then Lemma 6.29.2 holds. Assume that for some $m \geq k+1$ there exists a (μ, m)-valid $Z \supseteq_k X$ such that $q \in A(Z)$, $N \subseteq \overline{Z}$, and $Z^{>k+1}$ contains only μ-codewords. Hence, there exists some $y \in \Sigma^{3|q|+3}$ such that $0qy \in Z$.[6]

Let $Z' \stackrel{df}{=} Z^{\leq k+1}$. From Proposition 6.10.6 it follows that Z' is $(\mu, k+1)$-valid. Since $Z^{>k+1}$ contains only μ-codewords, we can apply Lemma 6.18 for $(\{0qy\}, \emptyset)$. We obtain a $(\mu, k+1)$-reservation (Y', N') for Z' such that $0qy \in Y'$, $\ell(Y' \cup N') \leq 2 \cdot |0qy| = 8 \cdot (|q|+1)$, and $Y' \subseteq Z \subseteq \overline{N'}$. Together with $N \subseteq \overline{Z}$, this implies

$$Y' \cap N = \emptyset. \tag{18}$$

Moreover, since $Y' \subseteq Z$, it holds that ${Y'}^{>k+1}$ contains only μ-codewords. It follows that (Y', N') must have been in L_A and has been removed during some iteration. This implies that during that iteration, $Y' \cap ({Y'_A}^{>k} \cup {N'_A}^{>k}) \neq \emptyset$ (by line 6). Moreover, by line 4, ${Y'_A}^{>k} \cup {N'_A}^{>k}$ is a subset of N when the algorithm stops. This implies $Y' \cap N \neq \emptyset$, which contradicts equation (18). This proves Lemma 6.29. □

PROPOSITION 6.32 (property P4). *Let $i \geq 1$ and let X be (μ, k)-valid. There exists an $l > k$ and a (μ, l)-valid $Y \supseteq_k X$ such that for all $Z \supseteq_l Y$, if $A(Z) \cap B(Z) = \emptyset$, then there exists a separator S of $(A(Z), B(Z))$ such that $F(Z) \neq L(M_i^S)$.*

Proof. By Lemma 6.17, we can assume that $k \equiv 2 \pmod 4$ and $64(k+10)^{3i} < 2^{k/2}$.

We describe the construction of S_A and S_B, which are sets of queries we reserve for $\overline{B(Y)}$ and $\overline{A(Y)}$, respectively. Let $S_A := A(X)$ and $S_B := B(X)$. We simulate the computation $M_i^{S_A}(0^{k+1})$ until we reach a query q_1 that belongs to neither S_A nor S_B. Note that $|q_1| \leq (k+1)^i \leq 2^{k/2-4} - 2$. From Lemma 6.29 we obtain some $N_1 \subseteq \Sigma^{>k}$ such that $\|N_1\| \leq (8 \cdot |q_1| + 10)^2$ and either property 6.29.1 or property 6.29.2 holds. If property 6.29.1 holds, then add q_1 to S_B; otherwise add q_1 to S_A. Now return the

[6] Actually, it even holds that $0qy \in Z - X$, but we do not need this explicitly in our argumentation. In order to see this, we assume that $0qy$ is in X. Then q is in $A(X)$ and $(\{0qy\}, \emptyset)$ is a (μ, k)-reservation for X. Therefore, $(\{0qy\}, \emptyset)$ is a $(\mu, k+1)$-reservation for every $(\mu, k+1)$-valid $Z \supseteq_k X$. Hence, $(\{0qy\}, \emptyset)$ is in L_A at the beginning of the algorithm. So it has been removed during the algorithm. But this is not possible since elements in L_A can only be removed in step 6, and there we remove only (Y_A, N_A) with $Y_A \cap \Sigma^{>k} \neq \emptyset$. This shows $0qy \in Z - X$.

answer of "$q_1 \in S_A$?" to the computation. We continue the simulation until we reach a query q_2 that belongs to neither S_A nor S_B. Again we apply Lemma 6.29, obtain the set N_2, and add q_2 either to S_A or to S_B. We continue the simulation until the computation stops. Let n be the number of queries that were added to S_A or S_B. Observe that $S_A \cap S_B = \emptyset$ at the end of our simulation.

Let $N \stackrel{df}{=} N_1 \cup \cdots \cup N_n \cup \{0^{4(k+1)^i+4}\}$. Then $\|N\| \leq (k+1)^i \cdot (8 \cdot (k+1)^i + 10)^2 + 1 \leq 2^{k/2}$. Hence there exists some $w \in \Sigma^{k+1} - N$. If the simulation accepts, then let $S' = \emptyset$; otherwise let $S' \stackrel{df}{=} \{w\}$. Since $S \subseteq \Sigma^{k+1}$ and $k+1 \equiv 3 \pmod 4$, we have $C(S') = D(S') = \emptyset$ and for all $n \geq 1$, $E_n(S') = \emptyset$. From Proposition 6.10.4, it follows that $Y' \stackrel{df}{=} X \cup S'$ is $(\mu, k+1)$-valid. Since $N \subseteq \Sigma^{>k}$ and $N \cap S' = \emptyset$, we have $N \subseteq \overline{Y'}$. Therefore, by Proposition 6.13.3, $(\emptyset, N)$ is a $(\mu, k+1)$-reservation for Y'. By Lemma 6.14, there exist an $l \geq 4(k+1)^i + 4$ and a (μ, l)-valid $Y \supseteq_{k+1} Y'$ such that $N \subseteq \overline{Y}$ and $Y^{>k+1}$ contains only μ-codewords. In particular, it holds that $l > k$ and $Y \supseteq_k X$.

CLAIM 6.33. *For every $Z \supseteq_l Y$ it holds that $S_A \subseteq \overline{B(Z)}$ and $S_B \subseteq \overline{A(Z)}$.*

Assume that $S_A \cap B(Z) \neq \emptyset$ for some $Z \supseteq_l Y$, and choose a $v \in S_A \cap B(Z)$. Since S_A contains only words of length $\leq (k+1)^i$, we obtain $v \in S_A \cap B(Z^{\leq 4(k+1)^i+4}) \subseteq S_A \cap B(Y)$. So v cannot belong to $A(Y)$ since $A(Y) \cap B(Y) = \emptyset$. In particular this means $v \in S_A - A(X)$; i.e., $v = q_j$ for a suitable j with $1 \leq j \leq n$. By our construction q_j was only added to S_A when property 2 of Lemma 6.29 holds. Remember that Y is (μ, l)-valid with $l > k$, $Y \supseteq_k X$, $N_j \subseteq N \subseteq \overline{Y}$, and $Y^{>k+1}$ contains only μ-codewords. Therefore, from property 6.29.2 it follows that $v = q_j \notin B(Y)$, which contradicts $v \in S_A \cap B(Y)$. This shows $S_A \subseteq \overline{B(Z)}$. By the symmetric argument we obtain $S_B \subseteq \overline{A(Z)}$. This proves Claim 6.33.

Consider any $Z \supseteq_l Y$ with $A(Z) \cap B(Z) = \emptyset$. Let $S \stackrel{df}{=} A(Z) \cup S_A$. Assume that S is not a separator of $(A(Z), B(Z))$. Since $A(Z) \subseteq S$, we must have $S \cap B(Z) \neq \emptyset$. Since $A(Z) \cap B(Z) = \emptyset$, this implies $S_A \cap B(Z) \neq \emptyset$. This contradicts Claim 6.33. So S is a separator of $(A(Z), B(Z))$. It remains to show $F(Z) \neq L(M_i^S)$.

By our construction, $0^{k+1} \in F(Y')$ if and only if $M_i^{S_A}(0^{k+1})$ rejects. Since $Z \supseteq_{k+1} Y'$, it holds that $0^{k+1} \in F(Z)$ if and only if $M_i^{S_A}(0^{k+1})$ rejects. Assume that there exists a query q that is answered differently in the computations $M_i^{S_A}(0^{k+1})$ and $M_i^S(0^{k+1})$ (take the first such query). Since $S_A \subseteq S$, we obtain $q \in S - S_A$, i.e., $q \in A(Z)$. If q is in $B(X)$, then q is in $B(Z) \subseteq \overline{S}$, which is not possible. So q is neither in S_A nor in $B(X)$, but q is asked in the computation $M_i^{S_A}(0^{k+1})$. It follows that $q = q_j$ for some j with $1 \leq j \leq n$, and during the construction we added q_j to S_B. So we have $q \in S_B \cap A(Z)$, which contradicts Claim 6.33. Therefore, $M_i^{S_A}(0^{k+1})$ accepts if and only if $M_i^S(0^{k+1})$ accepts. This shows $0^{k+1} \in F(Z)$ if and only if $M_i^S(0^{k+1})$ rejects, i.e., $F(Z) \neq L(M_i^S)$. □

This finishes the proof of Theorem 6.7. □

COROLLARY 6.34. *The oracle O_2 of Theorem 6.7 has the following additional properties:*

(i) $\mathrm{UP}^{O_2} \neq \mathrm{NP}^{O_2} \neq \mathrm{coNP}^{O_2}$ *and* $\mathrm{NPMV}^{O_2} \not\subseteq_c \mathrm{NPSV}^{O_2}$.

(ii) *Relative to O_2, no optimal propositional proof systems exist.*

(iii) *There exists a $\leq_{sm}^{pp}$-complete disjoint NP^{O_2}-pair (A, B) that is P^{O_2}-inseparable but symmetric.*

Proof. It is known that Conjecture 2.4 implies item (i) [ESY84, GS88, Sel94]. Relative to O_2, NP∩SPARSE does not have $\leq_m^{p,O_2}$-complete sets. Messner and Torán [MT98] proved that this implies that there are no optimal propositional proof systems. This shows (ii).

Since (A, B) is $\leq^{pp}_{sm}$-complete, it is symmetric. If (A, B) is P^{O_2}-separable, then every disjoint NP^{O_2}-pair is P^{O_2}-separable, and therefore symmetric. This contradicts item (ii) of Theorem 6.7. So (A, B) is P^{O_2}-inseparable. □

7. Relationship to optimal propositional proof systems. It is known that existence of optimal propositional proof systems implies existence of $\leq^{pp}_{m}$-complete disjoint NP-pairs. Messner and Torán [MT98] state that this result was communicated to them by Impagliazzo and Pitassi. Ben-David and Gringauze [BDG98] cite Razborov [Raz94] for this result. Köbler, Messner, and Torán [KMT03] cite Razborov, and they prove the stronger result that existence of optimal propositional proof systems implies existence of $\leq^{pp}_{sm}$-complete disjoint NP-pairs.[7] For the sake of completeness, we provide here a straightforward proof of the weaker result.

THEOREM 7.1. *If optimal propositional proof systems exist, then there is a* $\leq^{pp}_{m}$*-complete disjoint NP-pair.*

Proof. Let f be an optimal propositional proof system. We define the canonical pair [Raz94, Pud03] for this proof system, $(\mathrm{SAT}^*, \mathrm{REF}_f)$, where

$$\mathrm{SAT}^* = \{(x, 0^n) \mid x \in \mathrm{SAT}\}$$

and

$$\mathrm{REF}_f = \{(x, 0^n) \mid \neg x \in \mathrm{TAUT} \text{ and } \exists y[|y| \leq n \text{ and } f(y) = \neg x]\}.$$

Note that since f is polynomial-time computable, both SAT^* and REF_f are in NP. Also, for any n, if $(x, 0^n) \in \mathrm{SAT}^*$, then $x \in \mathrm{SAT}$, and if $(x, 0^n) \in \mathrm{REF}_f$, then $x \notin \mathrm{SAT}$. Therefore, these sets are disjoint, and so $(\mathrm{SAT}^*, \mathrm{REF}_f)$ is a disjoint NP-pair. We will prove that this pair is $\leq^{pp}_{m}$-complete.

Consider any other disjoint NP-pair (A, B). We will define a proof system $f_{A,B}$ using this pair. Assume that $A \leq^{p}_{m} \mathrm{SAT}$ via $g \in \mathrm{PF}$ and there is a polynomial $p(\cdot)$ and a polynomial-time predicate $R(\cdot, \cdot)$ such that $z \in B \Leftrightarrow \exists w, |w| \leq p(|z|), R(z, w)$.

$$f_{A,B}(y) = \begin{cases} \neg g(z) & \text{if } y = (z, w), \text{ where } |w| \leq p(|z|) \text{ and } R(z, w), \\ z & \text{if } y = (z, w), \text{ where } |w| > 2^{|z|} \text{ and } z \in \mathrm{TAUT}, \\ z \vee \neg z & \text{otherwise.} \end{cases} \tag{19}$$

We claim that $f_{A,B}$ is a proof system. First, note that for every $z \in \mathrm{TAUT}$, $f_{A,B}(z, w)$, for some $w, |w| > 2^{|z|}$, will output z in time polynomial in $|(z, w)|$. Also, since $A \cap B = \emptyset$ and g reduces A to SAT, $g(B) \subset \overline{\mathrm{SAT}}$. Therefore, for every $z \in B$ (i.e., for every z such that $R(z, w)$ for some $w, |w| \leq p(|z|)$), $g(z) \notin \mathrm{SAT}$. Therefore, $f_{A,B}$ outputs all possible tautologies and does not output anything that is not in TAUT. Also, since g is polynomial-time computable, so is $f_{A,B}$. It is therefore clear that $f_{A,B}$ is a proof system; since f is an optimal proof system, there is a polynomial $q(\cdot)$ such that for every tautology ϕ, and for every w such that $f_{A,B}(w) = \phi$, there is a $w', |w'| \leq q(|w|)$ and $f(w') = \phi$.

Now we define $h \in \mathrm{PF}$ such that $(A, B) \leq^{pp}_{m} (\mathrm{SAT}^*, \mathrm{REF}_f)$ via h. On input x, h outputs $(g(x), 0^{r(|x|)})$, where $r(\cdot)$ is some polynomial that we will fix later. If $x \in A$, then $g(x) \in \mathrm{SAT}$, and therefore $h(x) \in \mathrm{SAT}^*$.

On the other hand, for all $x \in B$, $g(x) \notin \mathrm{SAT}$, i.e., $\neg g(x) \in \mathrm{TAUT}$. Since $x \in B$, there exists $y = (x, w)$, where $|w| \leq p(|x|)$ such that $f_{A,B}(y) = \neg g(x)$. So,

[7] However, a forthcoming paper [GSS04] proves that there exist $\leq^{pp}_{sm}$-complete disjoint NP-pairs if and only if there exist $\leq^{pp}_{m}$-complete disjoint NP-pairs.

there is some y', $|y'| \leq q(|y|)$, such that $f(y') = \neg g(x)$. We choose r to be large enough so that $r(|x|) > |y'|$, and since q and p are polynomial, r can be chosen to be a polynomial as well. This shows that $x \in B$ implies $h(x) \in \mathrm{REF}_f$. Therefore, $(A, B) \leq_m^{pp} (\mathrm{SAT}^*, \mathrm{REF}_f)$; i.e., $(\mathrm{SAT}^*, \mathrm{REF}_f)$ is $\leq_m^{pp}$-complete. □

8. Conclusions. We partially summarize the import of the oracle results we obtained in this paper. Various implications have been known and/or are observed here for the first time. For several of these, our oracles demonstrate that the converses do not hold robustly. The following are convenient lists of these instances:

- Existence of optimal proof systems implies existence of $\leq_{sm}^{pp}$-complete NP-pairs [Raz94, KMT03]. Relative to oracle O_2, the converse is false.

Relative to both oracles O_1 and O_2, the converses of the following implications are false:

1. Nonexistence of $\leq_T^{pp}$-complete NP-pairs implies Conjecture 2.4 (observed in section 3).
2. Nonsymmetric implies P-inseparable (observed in section 5).
3. Nonexistence of $\leq_T^{pp}$-complete NP-pairs implies NP $\neq$ coNP (observed in section 3).
4. Nonexistence of $\leq_m^{pp}$-complete NP-pairs implies NP $\neq$ coNP (observed in section 3).

Acknowledgments. The authors thank Avi Wigderson for informing them of the paper by Ben-David and Gringauze [BDG98]. Also we thank the anonymous referees for their careful reading and thoughtful comments.

REFERENCES

[BGS75] T. Baker, J. Gill, and R. Solovay, *Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question*, SIAM J. Comput., 4 (1975), pp. 431–442.

[BDG98] S. Ben-David and A. Gringauze, *On the Existence of Propositional Proof Systems and Oracle-Relativized Propositional Logic*, Technical Report 5, Electronic Colloquium on Computational Complexity, 1998.

[BFFvM00] H. Buhrman, S. Fenner, L. Fortnow, and D. van Melkebeek, *Optimal proof systems and sparse sets*, in Proceedings of the 17th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1770, Springer-Verlag, Berlin, 2000, pp. 407–418.

[CR79] S. Cook and R. Reckhow, *The relative efficiency of propositional proof systems*, J. Symbolic Logic, 44 (1979), pp. 36–50.

[ESY84] S. Even, A. Selman, and J. Yacobi, *The complexity of promise problems with applications to public-key cryptography*, Inform. and Control, 61 (1984), pp. 159–173.

[FHOS97] S. Fenner, S. Homer, M. Ogihara, and A. Selman, *Oracles that compute values*, SIAM J. Comput., 26 (1997), pp. 1043–1065.

[FPS01] L. Fortnow, A. Pavan, and A. Selman, *Distributionally hard languages*, Theory Comput. Syst., 34 (2001), pp. 245–261.

[GW03] C. Glasser and G. Wechsung, *Relativizing function classes*, J. UCS, 9 (2003), pp. 34–50.

[GSS04] C. Glasser, A. Selman, and S. Sengupta, *Reductions between disjoint NP-pairs*, in Proceedings of the 19th IEEE Conference on Computational Complexity, IEEE Computer Society Press, Los Alamitos, CA, 2004, pp. 42–53.

[GS88] J. Grollmann and A. L. Selman, *Complexity measures for public-key cryptosystems*, SIAM J. Comput., 17 (1988), pp. 309–335.

[Gur83] Y. Gurevich, *Algebras of feasible functions*, in Proceedings of the 24th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1983, pp. 210–214.

[HY84] J. Hartmanis and Y. Yesha, *Computation times of NP sets of different densities*, Theoret. Comput. Sci., 34 (1984), pp. 17–32.

[HIS85] J. Hartmanis, N. Immerman, and V. Sewelson, *Sparse sets in NP − P: EXPTIME versus NEXPTIME*, Inform. and Control, 65 (1985), pp. 158–181.

[HJV93] L. Hemaspaandra, S. Jain, and N. Vereshchagin, *Banishing robust Turing completeness*, Internat. J. Found. Comput. Sci., 4 (1993), pp. 245–265.

[HS92] S. Homer and A. Selman, *Oracles for structural properties: The isomorphism problem and public-key cryptography*, J. Comput. System Sci., 44 (1992), pp. 287–301.

[KM00] J. Köbler and J. Messner, *Is the standard proof system for sat p-optimal?*, in Proceedings of the 20th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Lecture Notes in Comput. Sci. 1974, Springer-Verlag, Berlin, 2000, pp. 361–372.

[KMT03] J. Köbler, J. Messner, and J. Torán, *Optimal proof systems imply complete sets for promise classes*, Inform. and Comput., 184 (2003), pp. 71–92.

[KP89] J. Krajíček and P. Pudlák, *Propositional proof systems, the consistency of first order theories and the complexity of computations*, J. Symbolic Logic, 54 (1989), pp. 1063–1079.

[MT98] J. Messner and J. Torán, *Optimal proof systems for propositional logic and complete sets*, in Proceedings of the 15th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1373, Springer-Verlag, Berlin, 1998, pp. 477–487.

[PS02] A. Pavan and A. L. Selman, *Separation of NP-completeness notions*, SIAM J. Comput., 31 (2002), pp. 906–918.

[Pud86] P. Pudlák, *On the length of proofs of finitistic consistency statements in first order theories*, in Logic Colloquium '84, J. B. Paris et al., eds., North–Holland, Amsterdam, 1986, pp. 165–196.

[Pud03] P. Pudlák, *On reducibility and symmetry of disjoint NP-pairs*, Theoret. Comput. Sci., 1-3 (2003), pp. 323–339.

[Raz94] A. Razborov, *On Provably Disjoint NP-Pairs*, Technical Report TR94-006, Electronic Colloquium on Computational Complexity, 1994.

[Sel79] A. Selman, *P-selective sets, tally languages, and the behavior of polynomial-time reducibilities on NP*, Math. Systems Theory, 13 (1979), pp. 55–65.

[Sel88] A. Selman, *Promise problems complete for complexity classes*, Inform. and Comput., 78 (1988), pp. 87–98.

[Sel94] A. Selman, *A taxonomy of complexity classes of functions*, J. Comput. System Sci., 48 (1994), pp. 357–381.

SIAM J. COMPUT.
Vol. 33, No. 6, pp. 1417–1440

INCREMENTAL CLUSTERING AND DYNAMIC INFORMATION RETRIEVAL*

MOSES CHARIKAR†, CHANDRA CHEKURI‡, TOMAS FEDER§, AND RAJEEV MOTWANI¶

Abstract. Motivated by applications such as document and image classification in information retrieval, we consider the problem of clustering *dynamic* point sets in a metric space. We propose a model called *incremental clustering* which is based on a careful analysis of the requirements of the information retrieval application, and which should also be useful in other applications. The goal is to efficiently maintain clusters of small diameter as new points are inserted. We analyze several natural greedy algorithms and demonstrate that they perform poorly. We propose new deterministic and randomized incremental clustering algorithms which have a provably good performance, and which we believe should also perform well in practice. We complement our positive results with lower bounds on the performance of incremental algorithms. Finally, we consider the dual clustering problem where the clusters are of fixed diameter, and the goal is to minimize the number of clusters.

Key words. incremental clustering, dynamic information retrieval, minimum diameter clustering, agglomerative clustering, k-center, performance guarantee

AMS subject classifications. 68Q25, 68W40

DOI. 10.1137/S0097539702418498

1. Introduction.

We consider the following problem: as a sequence of points from a metric space is presented, efficiently maintain a clustering of the points so as to minimize the maximum cluster diameter. Such problems arise in a variety of applications, in particular in document and image classification for information retrieval. We propose a model called *incremental clustering* based primarily on the requirements for the information retrieval application, although our model should also be relevant to other applications. We begin by analyzing several natural greedy algorithms and discover that they perform rather poorly in this setting. We then identify some new deterministic and randomized algorithms with provably good performance. We complement our positive results with lower bounds on the performance of incremental algorithms. We also consider the dual clustering problem where the clusters are of fixed diameter, and the goal is to minimize the number of clusters. Before describing our results in any greater detail, we motivate and formalize our new model.

*Received by the editors November 25, 2002; accepted for publication (in revised form) April 2, 2004; published electronically August 27, 2004. A preliminary version of this paper appeared in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997.

http://www.siam.org/journals/sicomp/33-6/41849.html

†Department of Computer Science, Princeton University, Princeton, NJ 08544 (moses@cs.princeton.edu). The majority of this work was done while the author was at Stanford University and was supported by Rajeev Motwani's NSF award CCR-9357849. This author is currently supported by NSF ITR CCR-0205594, DOE award DE-FG02-02ER25540, NSF CAREER award CCR-0237113, and an Alfred P. Sloan fellowship.

‡Bell Labs, 600 Mountain Avenue, Murray Hill, NJ 07974 (chekuri@research.bell-labs.com). The majority of this work was done while the author was at Stanford University and was supported by Rajeev Motwani's NSF award CCR-9357849.

§E-mail: tomas@theory.stanford.edu.

¶Department of Computer Science, Stanford University, Stanford, CA 94305-9045 (rajeev@cs.stanford.edu). This author's work was supported by an Alfred P. Sloan Research fellowship, an IBM Faculty Partnership award, ARO MURI grant DAAH04-96-1-0007, and NSF Young Investigator award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

Clustering is used for data analysis and classification in a wide variety of applications [1, 20, 29, 35, 44]. It has proved to be a particularly important tool in *information retrieval* for constructing a taxonomy of a corpus of documents by forming groups of closely related documents [21, 24, 37, 44, 45, 47, 48]. For this purpose, a distance metric is imposed over documents, enabling us to view them as points in a metric space. The central role of clustering in this application is captured by the so-called *cluster hypothesis*: *documents relevant to a query tend to be more similar to each other than to irrelevant documents and hence are likely to be clustered together.* Typically, clustering is used to accelerate query processing by considering only a small number of representatives of the clusters, rather than the entire corpus. In addition, it is used for classification [19] and has been suggested as a method for facilitating browsing [16, 17].

The current information explosion, fueled by the availability of hypermedia and the World Wide Web, has led to the generation of an ever-increasing volume of data, posing a growing challenge for information retrieval systems to efficiently store and retrieve this information [50]. A major issue that document databases are now facing is the extremely high rate of update. Several practitioners have complained that existing clustering algorithms are not suitable for maintaining clusters in such a dynamic environment, and they have been struggling with the problem of updating clusters without frequently performing complete reclustering [7, 8, 9, 15, 45]. From a theoretical perspective, many different formulations are possible for this dynamic clustering problem, and it is not clear a priori which of these best addresses the concerns of the practitioners. After a careful study of the requirements, we propose the model described below.

Hierarchical agglomerative clustering. The clustering strategy employed almost universally in information retrieval is *hierarchical agglomerative clustering* (HAC); see [20, 44, 45, 47, 48, 49]. This is also popular in other applications such as biology, medicine, image processing, and geographical information systems. The basic idea is this: initially assign the n input points to n distinct clusters; repeatedly merge pairs of clusters until their number is sufficiently small. Many instantiations have been proposed and implemented, differing mainly in the rule for deciding which clusters to merge at each step. Note that HAC computes hierarchy trees of clusters (also called *dendograms*) whose leaves are individual points and whose internal nodes correspond to clusters formed by merging clusters at their children. A key advantage of these trees is that they permit refinement of responses to queries by moving down the hierarchy. Typically, the internal nodes are labeled with indexing information (sometimes called *conceptual information*) used for processing queries and in associating semantics with clusters (e.g., for browsing). Experience shows that HAC performs extremely well both in terms of efficiency and cluster quality. In the dynamic setting, it is desirable to retain the hierarchical structure while ensuring efficient update and high-quality clustering. An important goal is to avoid any major modifications in the clustering while processing updates, since any extensive recomputation of the index information will swamp the cost of clustering itself. The input size in typical applications is such that superquadratic time is impractical, and in fact it is desirable to obtain close to linear time.

A model for incremental clustering. Various measures of distance between documents have been proposed in the literature, but we will not concern ourselves with the details thereof; for our purposes, it suffices to note that these distance measures induce a metric space. Since documents are usually represented as high-dimensional vectors, we cannot make any stronger assumption than that of an arbitrary metric

space, although, as we will see, our results improve significantly in geometric spaces.

Formally, the *clustering* problem is as follows: given n points in a metric space $\mathcal{M}$, partition the points into k clusters so as to minimize the maximum cluster diameter. The *diameter* of a cluster is defined to be the maximum interpoint distance in it. Sometimes the objective function is chosen to be the maximum cluster radius. In Euclidean spaces, *radius* denotes the radius of the minimum ball enclosing all points in the cluster. To extend the notion of radius to arbitrary metric spaces, we first select a *center* point in each cluster, whereupon the radius is defined as the maximum distance from the center to any point in the cluster. We will assume the diameter measure as the default. We mention that there are several other measures of cluster quality that have been considered in the literature (e.g., sum of squares of distances to cluster centers, etc.). In this paper, we shall consider only the radius and diameter measures.

We define the *incremental clustering* problem as follows: for an update sequence of n points in $\mathcal{M}$, maintain a collection of k clusters such that as each input point is presented, either it is assigned to one of the current k clusters or it starts off a new cluster while two existing clusters are merged into one. We define the *performance ratio* of an incremental clustering algorithm as the maximum over all update sequences of the ratio of its maximum cluster diameter (or radius) to that of the optimal clustering for the input points.

Our model enforces the requirement that *at all times* an incremental algorithm should maintain a HAC for the points presented up to that time. As before, an algorithm is free to use any rule for choosing the two clusters to merge at each step. This model preserves all the desirable properties of HAC while providing a clean extension to the dynamic case. In addition, it has been observed that such incremental algorithms exhibit good paging performance when the clusters themselves are stored in secondary storage, while cluster representatives are preserved in main memory [42].

We have avoided labeling this model as the *online clustering problem* or referring to the performance ratio as a *competitive ratio* [34] for the following reasons. Recall that in an online setting, we would compare the performance of an algorithm to that of an adversary which knows the update sequence in advance but must process the points in the order of arrival. Our model has a stronger requirement for incremental algorithms, in that they are compared to adversaries which do not need to respect the input ordering, i.e., we compare our algorithms to optimal clusterings of the final point set, and no intermediate clusterings need be maintained. Also, online algorithms are permitted superpolynomial running times. In contrast, our model essentially requires polynomial-time approximation algorithms which are constrained to incrementally maintain a HAC. It may be interesting to explore the several different formulations of online clustering; for example, when the newly inserted point starts off a new cluster, we could allow the points of one old cluster to be redistributed among the remaining, rather than requiring that two clusters be merged together. The problem with such formulations is that they do not lead to HACs; moreover, they entail the recomputation of the index structures for *all clusters*, which renders the algorithms useless from the point of view of real applications. We note that since the incremental clustering model is more restrictive than the online model, our algorithms can also be viewed as online clustering algorithms; the competitive ratios are the same as the performance ratios that we prove here.

Previous work in static clustering. The closely related problems of clustering to minimize diameter and radius are also called *pairwise clustering* and the *k-center problem*, respectively [4, 30]. Both are NP-hard [25, 38], and in fact hard to approximate to within a factor of 2 for arbitrary metric spaces [4, 30]. For Euclidean spaces, clus-

tering on the line is easy [6], but in higher dimensions it is NP-hard to approximate to within factors close to 2, regardless of the metric used [22, 23, 27, 39, 40]. The *furthest point heuristic* due to Gonzalez [27] (see also Hochbaum and Shmoys [32, 33]) gives a 2-approximation in all metric spaces. This algorithm requires $O(kn)$ distance computations, and when the metric space is induced by shortest-path distances in weighted graphs, the running time is $O(n^2)$. Feder and Greene [22] gave an implementation for *Euclidean* spaces with running time $O(n \log k)$.

Clustering problems have been extensively studied in different communities from a variety of different perspectives. The optimization viewpoint of clustering formulates the problem as that of finding a solution that optimizes a certain objective function that measures cluster quality. In addition to the minimum diameter and radius measures described above, several other objective functions have also been considered in the literature. A lot of recent work has focused on the k-median objective [11, 36, 2]. Here the goal is to assign points to k centers such that the sum of distances of points to their centers is minimized. Other objectives that have been studied include the objective of minimizing the sum of all distances within each cluster [3, 18] and that of minimizing the sum of cluster diameters [14]. In addition to this, outlier formulations of clustering problems have also been studied [12]. Here the algorithm is allowed to discard a fraction of the input as outliers and is required to obtain a clustering solution that minimizes a given objective function on the remaining input points.

Overview of results. Our results for incremental clustering show that it is possible to obtain algorithms that are comparable to the best possible in the static setting, both in terms of efficiency and performance ratio. We begin in section 2 by considering natural greedy algorithms that choose clusters to merge based on some measure of the resulting cluster. We establish that greedy algorithms behave poorly by proving that a center-greedy algorithm has a tight performance ratio of $2k - 1$, and a diameter-greedy algorithm has a lower bound of $\Omega(\log k)$. It seems likely that greedy algorithms behave better in geometric spaces, and we discover some evidence in the case of the line. We show that diameter-greedy has performance ratio 2 for $k = 2$ on the line. This analysis suggests a variant of diameter-greedy, and this is shown to achieve ratio 3 for all k on the line. In section 3 we present the doubling algorithm and show that its performance ratio is 8, and that a randomized version has ratio 5.43. While the obvious implementation of these algorithms is expensive, we show that they can be implemented so as to achieve amortized time $O(k \log k)$ per update. These results for the doubling algorithm carry over to the radius measure. We also give a variant of this algorithm that is oblivious to the number of clusters k. We maintain a hierarchy from which, for any given k, at most k clusters can be obtained such that the diameter of the clusters is at most a factor of 8 times the optimal diameters for that value of k. Then, in section 4, we present the clique algorithm and show that it has performance ratio 6, and that a randomized version has ratio 4.33. While the clique algorithm may appear to dominate the doubling algorithm, this is not the case since the former requires computing clique partitions, an NP-hard problem, although it must be said in its defense that the clique partitions need only be computed in graphs with $k + 1$ vertices. While the performance ratio of the clique algorithm is 8 for the radius measure, improved bounds are possible for d-dimensional Euclidean spaces; specifically, we show that the radius performance ratio of the clique algorithm in $\Re^d$ improves to $4(1 + \sqrt{d/(2d + 2)})$, which is 6 for $d = 1$, and is asymptotic to 6.83 for large d. In section 5, we provide lower bounds for incremental clustering algorithms. We show that even for $k = 2$ and on the line, no deterministic or randomized algorithm can achieve a ratio better than 2.

TABLE 1.1

Summary of results: An asterisk indicates randomization. All algorithms have lower bounds on their performance ratios that match the upper bounds we establish.

Measure	Algorithm	Upper bound	Lower bound
Diameter			$1+\sqrt{2}$
	Center-greedy	$2k-1$	
	Diameter-greedy		$\Omega(\log k)$
	Doubling	8	
	Clique partition	6	
			$(2-\epsilon)^*$
	Doubling	5.437^*	
	Clique partition	4.33^*	
Radius			3
	Doubling	8	
	Clique partition	8	
			$(3-\epsilon)^*$
	Doubling	5.437^*	
	Clique partition	5.77^*	
Dual clust.			$\Omega(\frac{\log d}{\log\log\log d})$ in $\Re^d$
		$O(2^d d\log d)$ in $\Re^d$	$\Omega(2^d d\log d)$

We improve this lower bound to 2.414 for deterministic algorithms in general metric spaces. Finally, in section 6 we consider the dual clustering problem of minimizing the number of clusters of a fixed radius. Since it is impossible to achieve bounded ratios for general metric spaces, we focus on d-dimensional Euclidean spaces. We present an incremental algorithm that has performance ratio $O(2^d d\log d)$, and also provide a lower bound of $\Omega(\log d/\log\log\log d)$. See Table 1.1 for a summary of our results.

Many interesting directions for future research are suggested by our work. There are the obvious questions of improving our upper and lower bounds, particularly for the dual clustering problem. An important theoretical question is whether the geometric setting permits better ratios than metric spaces do. Our model can be generalized in many different ways. Depending on the exact application, we may wish to consider other measures of clustering quality, such as minimum variance in cluster diameter, and the sum of squares of the interpoint distances within a cluster. Then there is the issue of handling deletions which, though not important for our motivating application of information retrieval, may be relevant in other applications. Finally, there is the question of formulating a model for *adaptive clustering*, wherein the clustering may be modified as a result of queries and user feedback, even without any updates.

Recently, there has been considerable attention devoted to the streaming data model which was formalized and extensively studied after the appearance of the conference version of this paper. The streaming model is motivated by the goal of designing highly efficient algorithms for very large data sets. In this model, an algorithm is required to perform its computation in one pass or a few passes over the data while using very little memory. Our algorithms can be viewed as streaming algorithms for clustering problems where the goal is to minimize the maximum diameter or radius of the clusters produced. Subsequent to this work, other clustering objectives have also been studied in the streaming model, most notably the k-median objective [28, 13] and the sum of cluster diameters objective [14].

2. Greedy algorithms. We begin by examining some natural greedy algorithms. A *greedy incremental clustering algorithm* always merges clusters to minimize

some fixed measure. Our results indicate that such algorithms perform poorly.

DEFINITION 2.1. *The center-greedy algorithm associates a center for each cluster and merges the two clusters whose centers are closest. The center of the old cluster with the larger radius becomes the new center.*

It is possible to define variants of center-greedy based on how the centers of the clusters are picked, but we restrict ourselves to the above definition for reasons of simplicity and intuitiveness.

DEFINITION 2.2. *The diameter-greedy algorithm always merges those two clusters which minimize the diameter of the resulting merged cluster.*

We can establish the following lower bounds on the performance ratio of these two greedy algorithms.

THEOREM 2.3. *The center-greedy algorithm's performance ratio has a lower bound of* $2k-1$.

Proof. We first construct a graph $G=(V,E)$ which defines the metric space on which center-greedy has a performance ratio of $2k-1$. The vertex set of G is the union of $2k$ disjoint sets $S_0, S_1, \ldots, S_{2k-1}$ with a total of 2^k+k-1 vertices. The request sequence consists of the entire set of vertices $v_1, v_2, \ldots, v_{2^k+k-1}$. We use a complete binary tree $T=(V_T, E_T)$ whose leaves in a left to right order are the vertices $v_1, \ldots, v_{2^k}$ to describe our construction. Each internal node x of T has a binary *label* associated with it, denoted by $label(x)$. Let x_l and x_r be the left and right children of node x. We recursively define the labels of nodes of T as follows. The root of the tree is labeled with 0. If $label(x)=i$, we set $label(x_l)=i$ and $label(x_r)=1-i$. With each edge (x,y) of the tree we associate an integer weight of value $label(x)+label(y)-1$. Note that the weights belong to $\{-1,0,1\}$ by our labeling procedure. We now specify the sets $S_0, S_1, \ldots, S_{2k-1}$. A leaf v_i belongs to the set S_j if and only if the sum of the weights of the edges on the path from v_i to the root of T is equal to $j-k$. The vertices $v_{2^k+1}, \ldots, v_{2^k+k-1}$ belong to S_k. With each node x of T we also associate a vertex $v(x)$ of G as follows. If x is a leaf, we set $v(x)=x$; for an internal node x, we set $v(x)=v(x_r)$. For a node x of T let $po(x)=i$ if x is the ith vertex to be visited in a *post order* traversal of T, where we do not include the leaves in the traversal.

Now we specify the distances among the vertices of G. We first specify the edges present in G and then associate lengths with them. The other distances are induced by these edges. The edge set E is defined by the sets $S_0, \ldots, S_{2k-1}$ as follows:

$$\{(v_i,v_j) \mid v_i, v_j \in S_r, 0 \le r \le 2k-1\} \bigcup \{(v_i,v_j) \mid v_i \in S_r, v_j \in S_{r+1}, 0 \le r < 2k-1\}.$$

Edges between two vertices in the same set S_i are called *clique* edges. All the clique edges have length 1. The lengths of the other edges are defined as follows. We do a post order traversal of T and, as we process each internal node x of T, we set $d(v(x), v(x_l)) = 1-\epsilon_{po(x)}$ such that $1 \gg \epsilon_1 > \epsilon_2 > \cdots > \epsilon_{2^k-1} > 0$. By our placement of vertices in the sets S_i, we are guaranteed that $(v(x), v(x_l)) \in E$ for all x in T. We set to 1 any edge-length of an edge in E which is not already determined by the above process. See Figure 2.1 for an illustration of the construction.

Let C_x denote the cluster with $v(x)$ as the center and the leaves in the subtree rooted at x as the elements. Let $po^{-1}(i)$ denote the vertex x where $po(x)=i$. For $1 \le i \le 2^k-1$, let A_i be the set of clusters C_y such that y is the left child of some node on the path from $po^{-1}(i)$ to the root of T but does not itself lie on the path. We also include C_x in A_i, where $x=po^{-1}(i)$.

The lower bound is based on the following two claims.

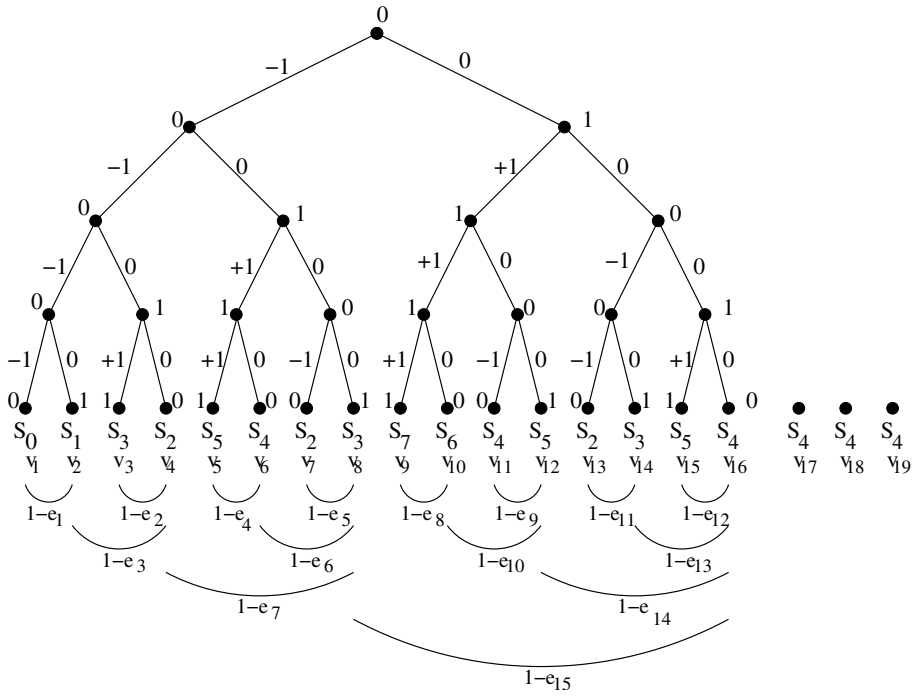


FIG. 2.1. *Illustration of the lower bound construction for center-greedy for $k = 4$. The instance consists of $2^k + k - 1 = 19$ vertices labeled $v_1, \ldots, v_{19}$. Each vertex belongs to one of the sets $S_0, \ldots, S_7$ as indicated. Vertex and edge labels defined on the tree structure are illustrated. The curved edges at the bottom of the figure indicate the additional edges added. (Note that distances within each S_i are all 1 and are not shown in the figure.) The curved edges also indicate the order in which clusters are merged by the center-greedy algorithm: cluster centers at the end points of the edge labeled $1 - \epsilon_1$ are first merged followed by centers at the end points of the edge labeled $1 - \epsilon_2$, then $1 - \epsilon_3$ and so on.*

CLAIM 2.4. *For $1 \le i \le 2k - 1$, A_i is the set of clusters of center-greedy which contain more than one vertex after the $k + i$ vertices $v_1, \ldots, v_{k+i}$ are given.*

Observe that $v_1 \in S_0$ and $v_{2^{k-1}+1} \in S_{2k-1}$, and the distance between them is at least $(2k - 1)(1 - \epsilon_1)$. From the above claim it follows that at the end of the request sequence, all the vertices $v_1, \ldots, v_{2^k}$ are in one cluster. Therefore, the diameter of this cluster is at least $(2k - 1)(1 - \epsilon_1)$.

CLAIM 2.5. *There is a k-clustering of G of diameter 1.*

The clustering which achieves the above diameter is $\{S_0 \cup S_1, \ldots, S_{2k-2} \cup S_{2k-1}\}$. This finishes the proof of the theorem. □

THEOREM 2.6. *The diameter-greedy algorithm's performance ratio is $\Omega(\log k)$, even on the line.*

Proof. We will prove a lower bound of $\Omega(\log k)$ for diameter-greedy on the real line. Let F_j be the jth Fibonacci number. Let k and r be such that $k = 2\sum_{i=1}^{r} F_i$. We will prove a lower bound of $r+1$ on diameter-greedy. To this end, we define sets of points A_{ij} for $1 \le i \le r$ and $1 \le j \le F_i$. Fix ϵ and δ such that $\epsilon < \delta \ll 1$. The set A_{ij} consists of the points p_{ij}, q_{ij}, r_{ij}, and s_{ij} sorted by their x-coordinates; the distances between them are as follows: $d(p_{ij}, q_{ij}) = 1 + \epsilon$, $d(q_{ij}, r_{ij}) = i$, and $d(r_{ij}, s_{ij}) = 1 + \delta$. Further, the distance between two distinct A_{ij}'s is set to ∞ (although any sufficiently large value will do). For $i = 1, \ldots, r$, let U_i, V_i, W_i, X_i, Y_i, and Z_i denote the following

clusters:

$$\begin{aligned}
U_i &= \cup_{j=1}^{F_i}\{\{p_{ij}, q_{ij}\}, \{r_{ij}, s_{ij}\}\},\\
V_i &= \cup_{j=1}^{F_i}\{\{q_{ij}\}, \{r_{ij}\}\},\\
W_i &= \cup_{j=1}^{F_i}\{\{p_{ij}\}, \{q_{ij}, r_{ij}\}\},\\
X_i &= \cup_{j=1}^{F_i}\{\{p_{ij}\}, \{q_{ij}, r_{ij}\}, \{s_{ij}\}\},\\
Y_i &= \cup_{j=1}^{F_i}\{\{p_{ij}, q_{ij}, r_{ij}\}, \{s_{ij}\}\},\\
Z_i &= \cup_{j=1}^{F_i}\{\{p_{ij}, q_{ij}, r_{ij}, s_{ij}\}\}.
\end{aligned}$$

Initially, the points given to diameter-greedy are the points q_{ij} and r_{ij} for each of the sets A_{ij}. Let P_i denote the sequence of requests $p_{i1}, \ldots, p_{iF_i}$ and let S_i denote the sequence $s_{i1}, \ldots, s_{iF_i}$. Define $K_i = P_1S_1P_2S_2 \ldots P_iS_i$. The request sequence is K_{r-1}. Note that $K_t = K_{t-1}P_tS_t$. We show inductively that the following invariant is maintained.

When the last element of K_t is received, diameter-greedy's $k+1$ clusters are

$$(\cup_{i=1}^{t-2} Z_i) \bigcup Y_{t-1} \bigcup X_t \bigcup (\cup_{i=t+1}^{r} V_i).$$

Since there are $k+1$ clusters, two of the clusters have to be merged and the algorithm merges two clusters in V_{t+1} to form a cluster of diameter $(t+1)$. Without loss of generality, we may assume that the clusters merged are $\{q_{(t+1)1}\}$ and $\{r_{(t+1)1}\}$.

Suppose that the situation is as described above at the end of the sequence K_t. We will show that the invariant holds at the end of the sequence K_{t+1}. Note that $K_{t+1} = K_tP_{t+1}S_{t+1}$. Observe that merging any two clusters in Y_{t-1} will increase their diameter to $(t+1+\epsilon+\delta)$. Merging any two clusters in X_t will increase their diameter to $(t+1+\epsilon)$. Because we start with the situation where there is a cluster $\{q_{(t+1)1}, r_{(t+1)1}\}$, the request $p_{(t+1)1}$ forces the formation of the cluster $\{q_{(t+1)2}, r_{(t+1)2}\}$, since any other merging results in a diameter of more than $t+1$. Thus, at the end of the sequence of requests P_{t+1}, we have $k+1$ clusters $(\cup_{i=1}^{t-2} Z_i) \bigcup Y_{t-1} \bigcup X_t \bigcup X_{t+1} \bigcup (\cup_{i=t+2}^{r} V_i)$.

Since there are $k+1$ clusters, diameter-greedy merges two clusters in X_t to form a cluster of diameter $(t+1+\epsilon)$. Thus, as we give points in S_{t+1}, it will form the clusters Y_t. Since $F_{t+1} > F_t$, after F_t requests in the sequence S_{t+1} all the clusters in X_t would have merged and diameter-greedy starts merging clusters in Y_{t-1} to form clusters Z_{t-1} of diameter $(t+1+\epsilon+\delta)$. Since $F_{t+1} = F_t + F_{t-1}$, at the end of the request sequence S_{t+1} it will have the $k+1$ clusters $(\cup_{i=1}^{t-1} Z_i) \bigcup Y_t \bigcup X_{t+1} \bigcup (\cup_{i=t+2}^{r} V_i)$. This shows that the invariant is maintained at the end of the request sequence K_{t+1}.

It is easy to verify that the invariant is true for the base case of $t = 3$. Thus, at the end of the request sequence K_{r-1}, the maximum diameter of the clusters of diameter-greedy is r. Since $k = 2\sum_{i=1}^{r} F_i = 2F_{r+2} - 2$, it follows that $r = \Omega(\log k)$. To finish the proof we observe that the optimal clusters are $\cup_{i=1}^{r-1} U_i \bigcup V_r$, and the diameter of these clusters is at most $1+\delta$. □

We now give a tight upper bound for the center-greedy algorithm. Note that for $k = 3$ it has ratio 5, but for larger k its performance is worse than that of the algorithms to be presented later.

THEOREM 2.7. *The center-greedy algorithm has performance ratio of $2k-1$ in any metric space.*

Proof. Suppose that a set P of n points is inserted. Let their optimal clustering be the partition $S = \{C_1, \ldots, C_k\}$, with d as the optimal diameter. We will show that the diameter of any cluster produced by center-greedy is at most $(2k-1)d$.

We define a graph G on the set S of the optimal clusters, where two clusters are connected by an edge if the minimum distance between them is at most d, where the distance between two clusters is the minimum distances between points in them. Consider the connected components of G. Note that two clusters in different connected components have a minimum distance strictly greater than d. We say that a cluster X intersects a connected component consisting of the optimal clusters $C_{i_1}, \ldots, C_{i_r}$ if X intersects $\cup_{j=1}^{r} C_{i_j}$.

We claim that at all times, any cluster produced by center-greedy intersects exactly one connected component of G. We prove this claim by induction over n. Suppose the claim is true before a new point p arrives. Initially, p is in a cluster of its own and center-greedy has $k+1$ clusters, each of which intersect exactly one connected component of G. Since there are $k+1$ cluster centers, two of them must be in the same optimal cluster. This implies that the distance between the two closest centers is at most d. If X_1 and X_2 are the clusters that center-greedy merges at this stage, the centers of X_1 and X_2 must be at most d apart. Hence, both clusters' centers must lie in the same connected component of G, say $\mathcal{C}$. By the inductive hypothesis, all points in X_1 and X_2 must be in $\mathcal{C}$. Hence, all points in the new cluster $X_1 \cup X_2$ must lie in $\mathcal{C}$, establishing the inductive hypothesis.

Since each cluster produced by center-greedy lies in exactly one connected component of G, the diameter is bounded by the maximum diameter of a connected component, which is at most $(2k-1)d$. ∎

For diameter-greedy in general metric spaces, we can only prove the following weak upper bound.

THEOREM 2.8. *For $k = 2$, the diameter-greedy algorithm has a performance ratio 3 in any metric space.*

Proof. Let d be the diameter in the optimal clustering. If the minimum distance between the two clusters (in the optimal clustering) is greater than d, then diameter-greedy keeps the two clusters separate and achieves the optimum. If the minimum distance between the two clusters is at most d, then all the points together form a cluster of diameter at most $3d$. ∎

In spite of the lower bounds for greedy algorithms, they may not be entirely useless since some variant may perform well in geometric spaces. We obtain some positive evidence in this regard via the following analysis for the line. The upper bounds given here should be contrasted with the lower bound of 2 for the line shown in section 5. The following definitions underlie the analysis.

DEFINITION 2.9. *Given a set S of n points in the line, a* t-partition *subdivides the interval between the first and last points of S into t subintervals whose end points are in S. The* t-diameter *of S is the minimum over all t-partitions of the maximum interval length in a t-partition of S. The* 1*-diameter is the diameter, while the* 2*-diameter is the radius of S where the center is constrained to be a point of S.*

We define the following family of algorithms based on the notion of the t-diameter.

DEFINITION 2.10. *The t-diameter-greedy algorithm merges those two clusters which minimize the t-diameter of the merged cluster. Note that* 1*-diameter-greedy is the same as diameter-greedy.*

A few preliminary results on the t-diameter-greedy algorithm can be found in the appendix.

3. The doubling algorithm. We now describe the doubling algorithm which has performance ratio 8 for incremental clustering in general metric spaces. The algorithm works in phases and uses two parameters α and β to be specified later, such that $\alpha/(\alpha-1) \leq \beta$. At the start of phase i, it has a collection of $k+1$ clusters $C_1, C_2, \ldots, C_{k+1}$ and a lower bound d_i on the optimal clustering's diameter (denoted by OPT). Each cluster C_i has a center c_i which is one of the points of the cluster. The following invariants are assumed at the start of phase i:

1. for each cluster C_j, the radius of C_j defined as $\max_{p \in C_j} d(c_j, p)$ is at most αd_i;
2. for each pair of clusters C_j and C_l, the intercenter distance $d(c_j, c_l) \geq d_i$;
3. $d_i \leq \text{OPT}$.

Each phase consists of two stages: the first is a *merging stage*, in which the algorithm reduces the number of clusters by merging certain pairs; the second is the *update stage*, in which the algorithm accepts new updates and tries to maintain at most k clusters without increasing the radius of the clusters or violating the invariants (clearly, it can always do so by making the new points into separate clusters). A phase ends when the number of clusters again exceeds k.

DEFINITION 3.1. *The* t-threshold graph *on a set of points* $P = \{p_1, p_2, \ldots, p_n\}$ *is the graph* $G = (P, E)$ *such that* $(p_i, p_j) \in E$ *if and only if* $d(p_i, p_j) \leq t$.

The merging stage works as follows. Define $d_{i+1} = \beta d_i$, and let G be the d_{i+1}-threshold graph on the $k+1$ cluster centers $c_1, c_2, \ldots, c_{k+1}$. The graph G is used to merge clusters by repeatedly performing the following steps while the graph is nonempty: pick an arbitrary cluster C_i in G and merge all its neighbors into it; make c_i the new cluster's center; and remove C_i and its neighbors from G. Let $C'_1, C'_2, \ldots, C'_m$ be the new clusters at the end of the merging stage. Note that it is possible that $m = k+1$ when the graph G has no edges, in which case the algorithm will be forced to declare the end of phase i without going through the update stage.

LEMMA 3.2. *The pairwise distance between cluster centers after the merging stage of phase* i *is at least* d_{i+1}.

LEMMA 3.3. *The radius of the clusters after the merging stage of phase* i *is at most* $d_{i+1} + \alpha d_i \leq \alpha d_{i+1}$.

Proof. Prior to the merging, the distance between two clusters which are adjacent in the threshold graph is at most d_{i+1}, and their radius is at most αd_i. Therefore, the radius of the merged clusters is at most

$$d_{i+1} + \alpha d_i \leq (1 + \alpha/\beta) d_{i+1} \leq \alpha d_{i+1},$$

where the last inequality follows from the choice that $\alpha/(\alpha-1) \leq \beta$. ∎

The update stage continues while the number of clusters is at most k. When a new point arrives, the algorithm attempts to place it in one of the current clusters without exceeding the radius bound αd_{i+1}: otherwise, a new cluster is formed with the update as the cluster center. When the number of clusters reaches $k+1$, phase i ends and the current set of $k+1$ clusters along with d_{i+1} are used as the input for the $(i+1)$st phase.

All that remains to be specified about the algorithm is the initialization. The algorithm waits until $k+1$ points have arrived and then enters phase 1 with each point as the center of a cluster containing just itself, and with d_1 set to the distance between the closest pair of points. It is easily verified that the invariants hold at the start of phase 1. The following lemma shows that the clusters at the end of the ith phase satisfy the invariants for the $(i+1)$st phase.

LEMMA 3.4. *The $k+1$ clusters at the end of the ith phase satisfy the following conditions:*

1. *The radius of the clusters is at most αd_{i+1}.*
2. *The pairwise distance between the cluster centers is at least d_{i+1}.*
3. *$d_{i+1} \leq$ OPT, where OPT is the diameter of the optimal clustering for the current set of points.*

Proof. We have $k+1$ clusters at the end of the phase since that is the terminating condition for a phase. From Lemma 3.3, the radius of the clusters after the merging stage is at most αd_{i+1}. When adding new points, the algorithm ensures that the radius bound is respected. From Lemma 3.2, the distance between the clusters centers after the merging stage is d_{i+1}, and a new cluster is created only if a request point is at least d_{i+1} away from all current cluster centers. Therefore, the cluster centers have pairwise distance at least d_{i+1}. Since at the end of the phase we have $k+1$ cluster centers that are d_{i+1} apart, the optimal clustering is forced to put at least two of them in the same cluster. It follows that OPT $\geq d_{i+1}$. □

We make the following observations. The algorithm ensures the invariant that $d_i \leq$ OPT at the start of phase i. The radius of the clusters during phase i is at most αd_{i+1}. Therefore, the performance ratio at any time during the phase is at most $2\alpha d_{i+1}/\text{OPT} \leq 2\alpha\beta d_i/\text{OPT} \leq 2\alpha\beta$. We choose $\alpha = \beta = 2$; note that this choice satisfies the condition that $\alpha/(\alpha-1) \leq \beta$. This leads to the following performance bound, which can be shown to be tight.

THEOREM 3.5. *The doubling algorithm has performance ratio 8 in any metric space.*

We give a simple example showing that our analysis of the doubling algorithm is tight. Here we assume that $k \geq 3$. The input sequence consists of $k+3$ points, $p_1, \ldots, p_{k+1}, p_{k+2}, p_{k+3}$, given in that order. The points $p_1, \ldots, p_{k+1}$ form a uniform metric space with distance 1. The points p_{k+2} and p_{k+3} are distance 4 from the points $p_1, \ldots, p_{k+1}$ and are distance 8 from each other. It is easy to ensure that both p_{k+2} and p_{k+3} will be added to the same cluster during the update stage of phase 1, and thus the diameter of the largest cluster formed by the doubling algorithm is 8. However, an optimum clustering can achieve diameter 1 by having p_{k+2} and p_{k+3} in their own clusters and the rest of the points in a single cluster of diameter 1.

An examination of the proof of the preceding theorem shows that the radius of the clusters is within a factor of 4 of the diameter of the optimal clustering, leading to the following result.

COROLLARY 3.6. *The doubling algorithm has performance ratio 8 for the radius measure.*

A simple modification of the doubling algorithm, in which we pick the new cluster centers by a simple left-to-right scan, improves the ratio to 6 for the case of the line.

While the obvious implementation of this algorithm appears to be inefficient, we can establish the following time bound, which is close to the best possible.

THEOREM 3.7. *The doubling algorithm can be implemented to run in $O(k \log k)$ amortized time per update.*

Proof. First of all, we assume that there is a black box for computing the distance between two points in the metric space in unit time. This is a reasonable assumption in most applications, and in any case even the static algorithms' analysis requires such an assumption. In the information retrieval application, the documents are represented as vectors and the black-box implementation will depend on the vector length as well as the exact definition of distance.

We now show how the doubling algorithm may be implemented so that the amortized time required for processing each new update is bounded by $O(k \log k)$. We maintain the edge lengths of the complete graph induced by the current cluster centers in a heap. Since there are at most k clusters the space requirement is $O(k^2)$. When a new point arrives, we compute the distance of this point to each of the current cluster centers, which requires $O(k)$ time. If the point is added to one of the current clusters, we are done. If, on the other hand, the new point initiates a new cluster, we insert into the heap edges labeled with the distances between this new center and the existing cluster centers. For accounting purposes in the amortized analysis, we associate $\log k$ credits with each inserted edge. We will show that it is possible to charge the cost of implementing the merging stage of the algorithm to the credits associated with the edges. This implies the desired time bound.

We can assume, without loss of generality, that the merging stage merges at least two clusters. Let t be the threshold used during the phase. The algorithm extracts all the edges from the heap which have length less than t. Let m be the number of edges deleted from the heap. The deletion from the heap costs $O(m \log k)$ time. The t-threshold graph on the cluster centers is exactly the graph induced by these m edges. It is easy to see that the procedure described to find the new cluster centers using the threshold graph takes time linear in the number of edges of the graph, assuming that the edges are given in the form of an adjacency list. Forming the adjacency list from the edges takes linear time. Therefore, the total cost of the merging phase is bounded by $O(m \log k + m) = O(m \log k)$ time. The credit of $\log k$ placed with each edge when it is inserted into the heap accounts for this cost, completing the proof. □

Finally, we describe a randomized doubling algorithm with significantly better performance ratio. The algorithm is essentially the same as before, the main change being in the value of d_1, which is the lower bound for phase 1. In the deterministic case we chose d_1 to be the minimum pairwise distance of the first $k+1$ points, say x. We now choose a random value r from $[1/e, 1]$ according to the probability density function $1/r$, set d_1 to rx, and redefine $\beta = e$ and $\alpha = e/(e-1)$. Similar randomization of doubling algorithms was used earlier in scheduling [41], and later in other applications [10, 26].

THEOREM 3.8. *The randomized doubling algorithm has expected performance ratio* $2e \approx 5.437$ *in any metric space. The same bound is also achieved for the radius measure.*

Proof. Let σ be the sequence of updates and let the optimal cluster diameter for σ be γx for some $\gamma \geq 1$; by the definition of x, the optimal value is at least x. Suppose we choose $d_1 = rx$ for some $r \in (0, 1]$. Let ρ_r be the radius of the clusters created for σ with this value of r. Using arguments similar to those in the proof of Theorem 3.5, we can show that ρ_r is at most $d_{i+1} + \alpha d_i = e^{i+1} d_1/(e-1)$, where i is the largest integer such that $d_i = e^{i-1} d_1 = e^{i-1} rx \leq \text{OPT} = \gamma x$. Let i^* be the integer such that $e^{i^*-1} \leq \gamma < e^{i^*}$ and $\delta = \gamma / e^{i^*}$. Then $\rho_r \leq \frac{re x \gamma}{(e-1)\delta}$ when $r > \delta$, and $\rho_r \leq \frac{re^2 x \gamma}{(e-1)\delta}$ when $r \leq \delta$. Let X_r^- and X_r^+ be indicator variables for the events $[r \leq \delta]$ and $[r > \delta]$, respectively. We claim that the expected value of ρ_r is bounded by

$$
\begin{aligned}
E[\rho_r] &\leq \int_{1/e}^{1} \frac{re\gamma x(eX_r^- + X_r^+)}{\delta r(e-1)} \mathrm{d}r \\
&= \text{OPT} \frac{e}{\delta(e-1)} \int_{1/e}^{1} (eX_r^- + X_r^+) \mathrm{d}r
\end{aligned}
$$

$$= \text{OPT}\frac{e\delta(e-1)}{\delta(e-1)}$$
$$= e\text{OPT}.$$

Therefore, the expected diameter is at most $2e\text{OPT}$. □

Remark 3.9. The randomized version of the doubling algorithm can be converted to a deterministic algorithm for the offline case which runs in $O(\frac{1}{\epsilon}nk^2)$ time and has a performance ratio of $4(1+\epsilon)$.

3.1. An oblivious clustering algorithm. In this section we describe an incremental hierarchical clustering algorithm that does not need an a priori bound on the number of clusters. From the hierarchy, given an integer k, we can obtain a k-clustering of the points such that the diameter of the clustering is at most a factor of 8 times the optimum diameter for the given value of k.

For convenience, assume that we have an a priori upper bound on the maximum distance between points, which we take to be 1. We maintain the current points in a tree, where every point is a vertex of the tree. For convenience, we assume that if a point is a vertex of the tree, then one of its children corresponds to the same vertex. The root, at depth 0, is a single vertex. The vertices at depth $i \geq 1$ are points within distance $1/2^{i-1}$ of their parents, and all the vertices at depth i are at a distance greater than $1/2^i$ from each other.

Initially, we have a single point at the root, which also occurs as the only child, the only child of this child, and so on; for conceptual clarity, we may assume that the depth of the tree is infinite. Suppose we have a tree representing the points at some stage, and a new point p comes in. Let i be the largest integer such that p is within $1/2^i$ of some point q at depth i. Then p is added at depth $i+1$, with q as its parent, with p as its only child, p as the only child of the child, and so on. So p is at depth $i+1$ with parent within distance $1/2^i$ as desired, and also the vertices representing p at depth $j \geq i+1$ are at a distance greater than $1/2^j$ of other points at depth j. See Figure 3.1 for an example.

It remains to indicate how we obtain the k clusters from the tree when k is given. Let i be the greatest depth containing at most k points. These are the k cluster centers. The subtrees of the vertices at depth i are the clusters. As points are added, the number of vertices at depth i increases; if it goes beyond k, then we change i to $i-1$, collapsing certain clusters; otherwise, the new point is inserted in one of the existing clusters.

THEOREM 3.10. *The algorithm that outputs the k clusters obtained from the tree construction has performance ratio 8 for the diameter measure and the radius measure.*

Proof. Suppose the optimal diameter is $1/2^{i+1} < d \leq 1/2^i$. Then points at depth i are in different clusters, so there are at most k of them. Let $j \geq i$ be the greatest depth containing at most k points. Then these are the cluster centers, and the vertices in the corresponding subtrees are at a distance of the root within $1/2^j + 1/2^{j+1} + 1/2^{j+2} + \cdots \leq 1/2^{i-1} < 4d$. Hence the radius of the clusters is at most $4d$, and thus the diameter is at most $8d$. This proof also implies that if the optimal radius is r, the radius of the clusters output is at most $8r$. □

The randomization technique used in the randomized doubling algorithm can also be applied here to get a better performance ratio (in expectation). The construction of the tree described above used distance threshold $1/2^i$ for depth i. Instead, we use distance threshold r/e^i for depth i, where r is chosen at random from the interval

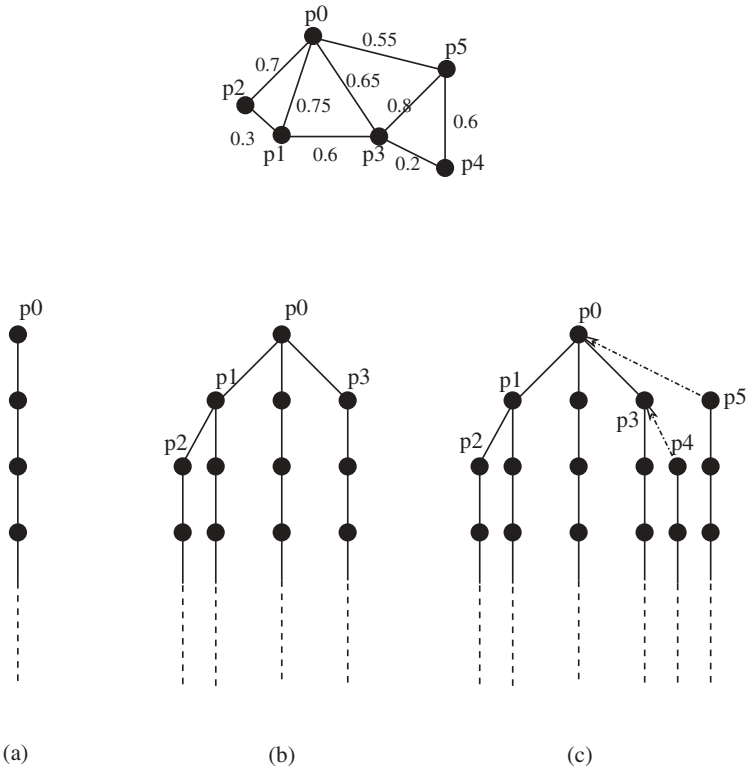


FIG. 3.1. *Illustration of the oblivious clustering algorithm. The points are numbered according to their order of arrival, and the distance between points is the minimum of* 1 *and the distance induced by the weighted graph shown.* (a) *Tree after first point* $p0$ *arrives.* (b) *Tree after four points.* (c) *Addition of* $p4$ *and* $p5$. *If* $k = 3$, *in* (b), *the clusters are centered around* $p0$, $p1$, *and* $p3$; *in* (c) *there is a single cluster around* $p0$.

$[1, e]$, according to the probability density function $1/r$. Note that the value r is chosen once at the beginning of the tree construction. By performing an analysis very similar to that for the randomized doubling algorithm, we can show that the expected diameter of the k-clustering obtained from the tree is at most $2e \approx 5.437$ times the optimal diameter. The same bound holds for the radius as well.

4. The clique partition algorithm. We now describe the clique algorithm, which has performance ratio 6. This does not totally improve upon the doubling algorithm since the new algorithm involves solving the NP-hard clique partition problem, even though it is only on a graph with $k + 1$ vertices. Finding a minimum clique partition is NP-hard even for graphs induced by points in the Euclidean plane [25], although it is in polynomial time for points on the line. Since the algorithm needs to solve the clique partition problem on graphs with $k + 1$ vertices, this may not be too inefficient for small k.

DEFINITION 4.1. *Given an undirected unweighted graph* $G = (V, E)$, *an* l*-clique partition is a partition of* V *into* $V_1, V_2, \ldots, V_l$ *such that for* $1 \leq i \leq l$, *the induced graph* $G[V_i]$ *is a clique. A* minimum clique partition *is an* l*-clique partition with the minimum possible value of* l.

The clique algorithm is similar to the doubling algorithm in that it also operates

in phases which have a merging stage followed by an update stage. The invariants maintained by the algorithm are different though. At the start of the ith phase we have $k+1$ clusters $C_1, C_2, \ldots, C_{k+1}$ and a value d_i such that

1. the radius of each cluster C_j is at most $2d_i$;
2. the diameter of each cluster C_j is at most $3d_i$;
3. $d_i \leq \text{OPT}$.

The merging works as follows. Let $d_{i+1} = 2d_i$. We form the minimum clique partition of the d_{i+1}-threshold graph G of the cluster centers. The new clusters are then formed by merging the clusters in each clique of the clique partition. We arbitrarily choose one cluster from each clique and make its center the cluster center of the new merged cluster. Let $C'_1, C'_2, \ldots, C'_{l_i}$ be the resulting clusters. In the rest of the phase we also need to know which old clusters merged to form each of the new clusters.

LEMMA 4.2. *After the merging stage, the radius of the new clusters is at most* $2d_{i+1}$ *and the diameter is at most* $3d_{i+1}$.

Proof. Let $C_{j_1}, C_{j_2}, \ldots, C_{j_{n_j}}$ be the clusters whose union is the new cluster C'_j and without loss of generality assume that the center of C_{j_1} was chosen to be the center of C'_j. Since the centers of $C_{j_1}, C_{j_2}, \ldots, C_{j_{n_j}}$ induce a clique in the d_{i+1}-threshold graph, the distance from the new center to each of the old cluster centers is at most d_{i+1}. The radius of each of $C_{j_1}, C_{j_2}, \ldots, C_{j_{n_j}}$ is at most $2d_i$. Therefore it follows that the new radius is at most $d_{i+1} + 2d_i \leq 2d_{i+1}$ and the diameter is at most $2d_i + d_{i+1} + 2d_i \leq 3d_{i+1}$. ∎

During the update phase, a new point p is handled as follows. Let the current number of clusters be m, where $l_i \leq m \leq k$. Recall that $C'_1, C'_2, \ldots, C'_{l_i}$ are the clusters formed during the merging stage. If there exists j such that $j > l_i$ and $d(p, c'_j) \leq d_{i+1}$, or if $j \leq l_i$ and $d(p, c_{j_s}) \leq d_{i+1}$ where C_{j_s} is a cluster which merged to form C'_j, add p to the cluster C'_j. If no such j exists, make a new cluster with p as the center. The phase ends when the number of clusters exceeds $k+1$, or if there are $k+1$ clusters at the end of the merging phase.

The following lemmas show that the clusters at the end of phase i satisfy the invariants for phase $i+1$.

LEMMA 4.3. *The radius of the clusters at the end of the phase* i *is at most* $2d_{i+1}$ *and the diameter of the clusters is at most* $3d_{i+1}$.

Proof. From Lemma 4.2, the radius of the clusters at the end of the merging stage of phase i is at most $2d_{i+1}$ and the diameter is at most $3d_{i+1}$. We now argue that these bounds are maintained during the update stage. If a point p is added to a cluster C_j, $j > l_i$, it satisfies the condition $d(p, c'_j) \leq d_{i+1}$; hence the radius of such a cluster is at most d_{i+1}, and hence the diameter is at most $2d_{i+1}$. If p is added to a cluster C'_j, $j \leq l_i$, then p satisfies the condition $d(p, c_{j_s}) \leq d_{i+1}$ where c_{j_s} is the center of a cluster that was merged to form the cluster C'_j. The center c'_j of C'_j is at a distance at most d_{i+1} from the centers of all the clusters that were merged to form C'_j. Therefore p is at a distance $2d_{i+1}$ from c'_j. This shows the radius bound. For the diameter bound, consider any two points p and q in the cluster C'_j. There must be cluster centers c_{j_s} and c_{j_t} of clusters that merged to form C'_j such that $d(p, c_{j_s}) \leq d_{i+1}$ and $d(q, c_{j_t}) \leq d_{i+1}$. The diameter bound follows since $d(c_{j_s}, c_{j_t}) \leq d_{i+1}$. ∎

LEMMA 4.4. *At the end of phase* i, $d_{i+1} \leq \text{OPT}$.

Proof. Suppose $d_{i+1} > \text{OPT}$. Let $S = \{c_1, c_2, \ldots, c_{k+1}\}$ be the cluster centers at the beginning of the phase i. Note that the centers $c'_1, \ldots, c'_{l_i}$ belong to S. Let $S' = \{c'_j \mid j > l_i\}$ be the set of cluster centers of the clusters which are formed

in phase i after the merging stage. Since each center c'_j in S' started a new cluster, $d(c'_j, p) > d_{i+1}$ for all $p \in S \cup S' - \{c'_j\}$. Therefore, in the optimal solution, each center in S' is in a cluster which contains no center from S. This implies that the centers in S are contained in at most $l_i - 1$ clusters of diameter d_{i+1}. This is a contradiction since l_i was the size of the minimum clique partition of the d_{i+1}-threshold graph on S. □

THEOREM 4.5. *The clique algorithm has performance ratio* 6 *in any metric space.*

Proof. The diameter of the clusters during phase i is at most $3d_{i+1}$, and we maintain the invariant that $d_i \leq \text{OPT}$ at the start of the phase. Therefore the performance ratio of the algorithm is at most $3d_{i+1}/d_i \leq 6$. □

The analysis of the clique algorithm is tight, as shown by the following example. We assume that $k \geq 4$. The sequence consists of $k+3$ points $p_1, \ldots, p_{k+1}, p_{k+2}, p_{k+3}$. The first k points form a uniform metric space with distance 1. The point p_{k+1} is at a distance 2 from p_2 and distance 1 from p_i for $2 \leq i \leq k$. The point p_{k+2} has the following properties: $d(p_{k+2}, p_1) = 2$, $d(p_{k+2}, p_i) = 3$ for $2 \leq i \leq k$, $d(p_{k+2}, p_{k+1}) = 4$, and $d(p_{k+2}, p_{k+3}) = 6$. The point p_{k+3} has the following properties: $d(p_{k+3}, p_1) = 4$, $d(p_{k+3}, p_i) = 3$ for $2 \leq i \leq k$, $d(p_{k+3}, p_{k+1}) = 2$, and $d(p_{k+3}, p_{k+2}) = 6$. After the first $k+1$ points are given the algorithm enters phase 1 with $d_1 = 1$ and $d_2 = 2$. In the merging stage the first $k+1$ points are merged into one single cluster since they form a clique in the d_2-threshold graph. It is easy to see that points p_{k+2} and p_{k+3} will be added to this cluster in the update stage, and the diameter of this cluster is seen to be 6. There is, however, an optimal clustering that achieves diameter 1 for all clusters: p_{k+2} and p_{k+3} are in their own clusters, p_1 and p_{k+1} are in different clusters, and the rest of the points can go either into the cluster containing p_1 or into the one containing p_{k+1}.

Since the radius of the clusters is within 4 of the optimal diameter, we obtain the following corollary.

COROLLARY 4.6. *The clique algorithm has performance ratio* 8 *in any metric space for the radius measure.*

As in the case of the doubling algorithm, we can use randomization to improve the bound. Let x be the minimum distance among the first $k+1$ points. The randomized algorithm sets $d_1 = rx$ in phase 1 of the deterministic algorithm, where r is chosen from $[1/2, 1]$ according to the probability density function $\frac{1}{r \ln 2}$. The analysis is similar to that of Theorem 3.8 and we omit the details.

THEOREM 4.7. *The randomized clique algorithm has performance ratio* $\frac{3}{\ln 2} \approx 4.33$ *in any metric space.*

COROLLARY 4.8. *The randomized clique algorithm has performance ratio* $\frac{4}{\ln 2} \approx 5.77$ *for the radius measure in any metric space.*

The special structure of the clusters in the clique algorithm can be used to show that the performance ratio for the radius measure is better than 8 for the geometric case. This is based on the following result in geometry, known as Jung's theorem (see [5, pp. 84–85]).

PROPOSITION 4.9. *Any convex set in* R^d *of diameter at most* 1 *can be circumscribed by a sphere of radius* $r_d = \sqrt{d/(2d+2)}$.

THEOREM 4.10. *The clique algorithm has performance ratio* $4(1 + r_d)$ *for the* radius *measure in* R^d. *This implies performance ratio* 6 *for* $d = 1$, 6.3 *for* $d = 2$, *and* 6.83 *asymptotically for large* d.

Proof. Let a_i be the maximum radius of the clusters of the algorithm in phase

i and let a_i^* be the radius achievable by an optimal clustering. We claim that $a_i \leq d_{i+1}(1+r_d) = 2d_i(1+r_d)$. Trivially, we have that $a_i^* \geq \text{OPT}_i/2$, where OPT_i is the optimal diameter achievable for points at the start of phase i. From the invariants of the algorithm we have that $d_i \leq \text{OPT}_i$, and hence we get that $a_i \leq 4a_i^*(1+r_d)$.

Now we prove the claim by induction. Assuming that the claim is true in phase $i-1$, we prove that it is true in phase i. In phase i let $C_1', C_2', \ldots, C_{l_i}'$ be clusters formed in the merging stage. It is easy to see that any new cluster formed during the update stage has radius at most d_{i+1}; hence we can focus on the clusters $C_1', C_2', \ldots, C_{l_i}'$. Without loss of generality consider C_1', which is formed by the merging of clusters $C_1, C_2, \ldots, C_j$ from the start of the phase. Let c_i be the center of C_i, $1 \leq i \leq j$. Since the clusters were merged, it follows that the diameter of the point set $\{c_1, c_2, \ldots, c_j\}$ is at most d_{i+1}. Hence their radius, by Proposition 4.9, is at most $d_{i+1}r_d$. From the invariants at the start of the phase, $d(p, c_i) \leq 2d_i = d_{i+1}$ for $p \in C_i$. Further, any new point added to the cluster C_1' is at most a distance of d_{i+1} from a point in $\{c_1, c_2, \ldots, c_j\}$. It follows that the radius of the cluster C_1' throughout the phase is at most $d_{i+1}r_d + d_{i+1}$, which proves the claim. ∎

5. Lower bounds. We present some lower bounds on the performance of incremental clustering. The lower bounds apply to both diameter and radius measures, but our proofs are given for the diameter case. The following theorem shows that even for the simplest geometric space, we cannot expect a ratio better than 2.

THEOREM 5.1. *For $k = 2$, there is a lower bound of 2 and $2 - 1/2^{k/2}$ on the performance ratio of deterministic and randomized algorithms, respectively, for incremental clustering on the line.*

Proof. Start with the three points $1, 2, 3$. Two consecutive points are necessarily merged, say 2 and 3. Add a new point at 4. Then 1 or 4 is merged with $[2, 3]$. This gives diameter 2, while the optimum is 1. In fact, this construction can be repeated to obtain n points $1, 2, \ldots, n$ clustered into 1 and $[2, n]$.

For the randomized lower bound, consider the instance in the preceding paragraph. To convert this to a randomized lower bound, the adversary places the fourth point at 0 or 4 with equal probability, and now the algorithm has probability $1/2$ of creating the wrong cluster (that is, with diameter 2). This gives a lower bound of 1.5. For general k, the algorithm makes $k/2$ copies of the above scenario far enough from each other that the above analysis applies locally. The probability that the algorithm succeeds in creating clusters of diameter 1 on all $k/2$ copies is $2^{-k/2}$. This implies a lower bound of $2 - 2^{-k/2}$. ∎

In the case of general metric spaces, we can establish a stronger lower bound.

THEOREM 5.2. *There is a lower bound of $1+\sqrt{2} \approx 2.414$ on the performance ratio of any deterministic incremental clustering algorithm for arbitrary metric spaces.*

Proof. Consider a metric space consisting of the points p_{ij}, $1 \leq i, j \leq 4$, $i \neq j$. The distances between the points are the shortest path distances in the graph with the following distances specified: $d(p_{ij}, p_{ji}) = 1$, and $d(p_{ij_1}, p_{ij_2}) = \sqrt{2}$ for $j_1 \neq j_2$. Let $B_i = \{p_{ij} \mid 1 \leq j \leq 4, i \neq j\}$. Note that the sets B_i, $1 \leq i \leq 4$, partition the metric space into 4 clusters of diameter $\sqrt{2}$. See Figure 5.1. Let A be any deterministic algorithm for the incremental clustering problem. Let $k = 6$. Consider the clusters produced by A after it is given the 12 points p_{ij} described above.

Case 1. Suppose the maximum diameter of A's clusters is 1. Then A's clusters must be the 6 sets $\{p_{ij}, p_{ji}\}$. Now the adversary gives a point q such that $d(q, p_{ij}) = 10$ for $1 \leq i, j \leq 4$. The optimal clustering is $\{q\}$ and the sets B_1, B_2, B_3, B_4. The optimal diameter is $\sqrt{2}$. We claim that the maximum diameter of A is at least $2 + \sqrt{2}$. If the

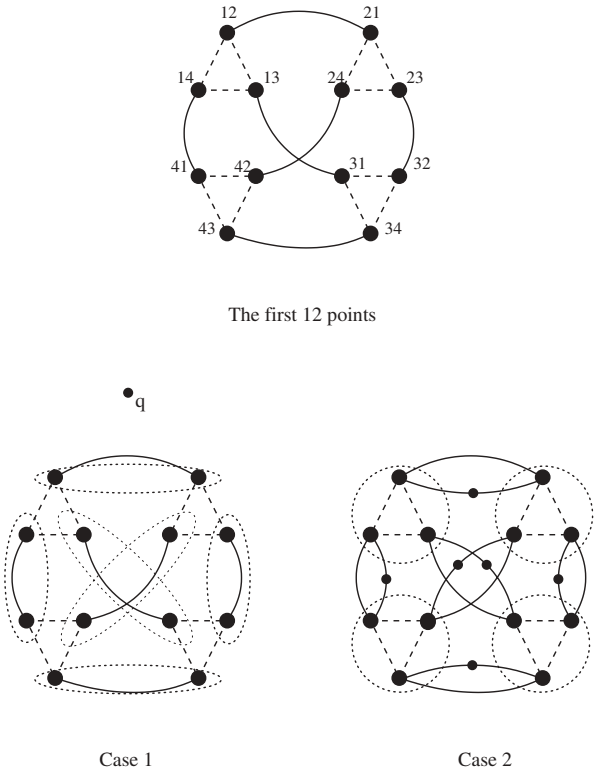


FIG. 5.1. *Lower bound instance. Solid and dashed lines show edges of length* 1 *and* $\sqrt{2}$*, respectively. Dotted ellipses show potential clusters. New points inserted after the first batch of* 12 *points are shown as smaller circles.*

cluster that contains q contains any other point, then our claim is clearly true. If on the other hand the cluster that contains q does not contain any other point, A must have merged two of its existing clusters. Then the maximum diameter of A's resulting clusters is at least $2+\sqrt{2}$. Thus the performance ratio of A is at least $1+\sqrt{2}$.

Case 2. Suppose the maximum diameter of A's clusters is greater than 1. Then some cluster of A contains 2 points which are at least distance $\sqrt{2}$ apart. Let these points be p_{wx} and p_{yz}, $(w,x) \neq (z,y)$. Now the adversary gives 6 points r_{ij}, $1 \leq i < j \leq 4$, such that $d(r_{ij}, p_{ij}) = d(r_{ij}, p_{ji}) = 1$. The optimal clustering consists of the 6 sets $\{r_{ij}, p_{ij}, p_{ji}\}$. The optimal diameter is 1. We claim that the maximum diameter of A's clusters must be at least $1+\sqrt{2}$. Note that $d(r_{i_1 j_1}, p_{i_2 j_2}) \geq 1+\sqrt{2}$ for $(i_2, j_2) \neq (i_1, j_1)$, $(i_2, j_2) \neq (j_1, i_1)$. Also $d(r_{i_1 j_1}, r_{i_2 j_2}) \geq 2+\sqrt{2}$ for $(i_1, j_1) \neq (i_2, j_2)$. If A puts any two r_{ij} in the same cluster, our claim is clearly true. If all the r_{ij} are in separate clusters, each of the 6 clusters must contain one of them. Also one of the 6 clusters, say C, must contain both the points p_{wx} and p_{yz}. Then C must have diameter at least $1+\sqrt{2}$, since the r_{ij} in C must be at a distance at least $1+\sqrt{2}$ from one of p_{wx} and p_{yz}. Hence the performance ratio of A is at least $1+\sqrt{2}$.

This proves a lower bound of $1+\sqrt{2}$ on the performance ratio of any deterministic incremental algorithm. □

Finally, we can improve the randomized lower bound slightly for the case of arbitrary metric spaces.

THEOREM 5.3. *For any $\epsilon > 0$ and $k \geq 2$, there is a lower bound of $2 - \epsilon$ on the performance ratio of any randomized incremental algorithm.*

Proof. We use Yao's technique and show a lower bound on deterministic algorithms on an appropriately chosen distribution. Let A be a deterministic algorithm for incremental clustering. The distribution on inputs is as follows. Initially, the adversary provides n points $P_1, P_2, \ldots, P_n$ such that the distance between any two of them is 1. Then the adversary partitions the n points into k disjoint sets $S_1, S_2, \ldots, S_k$ at random, such that all partitions are equally likely. Finally the adversary provides k points $Q_1, Q_2, \ldots, Q_k$, such that $d(Q_i, P_j) = 1$ if $P_j \in S_i$, $d(Q_i, P_j) = 2$ if $P_j \notin S_i$, $d(Q_i, Q_j) = 3$. Now, the diameter of the optimal solution for any input in the distribution is 1, obtained by constructing the k clusters $S_i \cup \{Q_i\}$. However, the incremental algorithm can produce a clustering with diameter 1 only if the clusters it produces after it sees points $P_1, P_2, \ldots, P_n$ are precisely the sets S_i (selected at random by the adversary). Let $X_k(n)$ be the number of ways to partition the n points into k sets. Then the probability that the incremental algorithm produces a clustering of diameter 1 is at most $p = 1/X_k(n)$. With probability at least $1 - p$, the incremental algorithm produces a clustering of diameter at least 2. Thus the expected value of the diameter of the clustering produced is at least $2 - p$. Hence the expected value of the performance ratio is at least $2 - p$. By choosing n suitably large, $X_k(n)$ can be made arbitrarily large, and hence p can be made arbitrarily small, in particular smaller than ϵ for any fixed $\epsilon > 0$. □

For the radius measure we have the following theorem.

THEOREM 5.4. *For the radius measure, no deterministic incremental clustering algorithm has a performance ratio better than 3 and no randomized algorithm has a ratio better than $3 - \epsilon$ for any fixed $\epsilon > 0$.*

Proof. We first consider the randomized case. The instances are very similar to the ones used in the proof of Theorem 5.3. The only difference is that $d(Q_i, P_j) = 0.5$ if $P_j \in S_i$ and $d(Q_i, P_j) = 1.5$ if $P_j \notin S_i$. The optimal clusters are $S_i \cup \{Q_i\}$ for $1 \leq i \leq k$ and each of them has a radius of 0.5. Any other clustering has radius at least 1.5 and the algorithm has a probability of $(1 - 1/X_k(n))$ of having such a cluster.

The instance used above can be adapted easily to show a bound of 3 for the deterministic case; we leave the details to the reader. □

6. Dual clustering. We now consider the dual clustering problem: for a sequence of points $p_1, p_2, \ldots, p_n \in \Re^d$, cover each point with a unit ball in $\Re^d$ as it arrives, so as to minimize the total number of balls used. In the static case, this problem is NP-complete and a PTAS is achievable in any fixed dimension [31]. We note that in general metric spaces, it is impossible to achieve any bounded ratio (for example, consider the uniform metric space).

Our algorithm's analysis is based on a theorem from combinatorial geometry called Roger's theorem [46] (see also [43, Theorem 7.17]), which says that R^d can be covered by any convex shape with covering density $O(d \log d)$. Since the volume of a radius 2 ball is 2^d times the volume of a unit-radius ball, the number of balls needed to cover a ball of radius 2 using balls of unit radius is $f(d) = O(2^d d \log d)$. We first describe an incremental algorithm which has performance ratio $f(d)$. We also establish an asymptotic lower bound of $\Omega(\frac{\log d}{\log \log \log d})$; for $d = 1$ and 2, our proof yields lower bounds of 2 and 4, respectively.

THEOREM 6.1. *For the dual clustering problem in $\Re^d$, there is an incremental algorithm with performance ratio $O(2^d d \log d)$.*

Proof. Our incremental algorithm maintains a set $\mathcal{C}$ of *centers* which is a subset

of the points that have arrived so far; initially, $\mathcal{C} = \emptyset$. Define the *range* $R(p)$ of a center p to be the sphere of radius 2 about p. For any two centers p_1 and p_2, we ensure that $d(p_1, p_2) > 2$. Associated with each center p is a set of points $\Gamma(p)$ called the *neighbors* of p. For convenience, we assume that $p \in \Gamma(p)$. We ensure that all neighbors of p lie in $R(p)$. When a new point x is received, if $x \in R(p)$ for some center p, we add x to $\Gamma(p)$, breaking ties arbitrarily. If no such center exists, x must be at a distance greater than 2 from all the existing centers. In this case, we make x a new center and set $\Gamma(x) = \{x\}$.

From Roger's theorem on packing density, a sphere of radius 2 in $\Re^d$ can be covered by $f(d) = O(2^d d \log d)$ spheres of radius 1. When a new center p is created, we fix a set of spheres $S_1(p), S_2(p), \ldots, S_{f(d)}(p)$ which cover $R(p)$. Whenever a point x is added to $\Gamma(p)$, if it is not already covered by some previously placed sphere, we add the sphere $S_r(p)$, where r is any value such that $x \in S_r(p)$. Note that such a sphere must exist as $x \in R(p)$ and the spheres $S_1(p), S_2(p), \ldots, S_{f(d)}(p)$ cover $R(p)$ completely.

Since no two centers can be covered by a sphere of unit radius, any solution must use a separate sphere to cover each center. Hence, the number of centers is a lower bound for the number of spheres used by the optimal offline algorithm. For each center p, the incremental algorithm uses at most $f(d)$ spheres to cover the points in $\Gamma(p)$. Hence, the performance ratio of the incremental algorithm is bounded by $f(d) = O(2^d d \log d)$. □

THEOREM 6.2. *For the dual clustering problem in $\Re^d$, any incremental algorithm must have performance ratio $\Omega(\frac{\log d}{\log \log \log d})$.*

Proof. The idea is as follows. At time t, when t points have been given by the adversary, it will be the case that the points $p_1, \ldots, p_t$ can be covered by a ball of radius $R_t < 1$. Then the adversary will find a point p_{t+1} lying outside the t unit balls laid down by the algorithm so as to minimize the radius R_{t+1} of the ball required to cover all $t + 1$ points and present that as a request. The game terminates when, at some time $k+1$, we have for the first time that $R_{k+1} > 1$. Clearly, k is a lower bound on the performance ratio since the points $p_1, \ldots, p_k$ can be covered by a ball of radius $R_k \leq 1$, and the algorithm has used k balls up to that point. It remains to analyze the worst-case growth rate of R_t as a function of t. Note that $R_1 = 0$ and $R_2 = 1/2$.

Let α_d denote the volume of a unit ball in $\Re^d$. At time t, let D_t be any ball of radius (at most) R_t that covers the points $p_1, \ldots, p_t$. For some δ_t to be specified later, define the ball D_t^* as a ball with the same center as D_t and with radius $R_t + \delta_t$. We will choose δ_t such that the volume of D_t^* is at least $t\alpha_d$, implying that the current t unit balls placed by the algorithm cannot cover the entire volume of D_t^*. This would imply that there is a choice of a point p_{t+1} inside D_t^* which is not covered by the current t balls. It is also clear that the new set of $t + 1$ points now can be covered by a ball of radius at most $R_t + \delta_t/2$, implying that

$$R_{t+1} \;=\; R_t + \frac{\delta_t}{2}.$$

Determining the value of δ_t is easy, since we have the inequality

$$\alpha_d (R_t + \delta_t)^d \;>\; t\alpha_d$$

from the requirement that the ball D_t have volume equal to that of t unit balls. Now let $R_t = 1 - \epsilon_t$. Substituting in the above equations, we obtain that $\delta_t = 2(\epsilon_t - \epsilon_{t+1})$

and hence $R_t + \delta_t = 1 + \epsilon_t - 2\epsilon_{t+1}$. Therefore

$$\alpha_d(1 + \epsilon_t - 2\epsilon_{t+1})^d \;>\; t\alpha_d,$$

which implies that

$$\ln(1 + \epsilon_t - 2\epsilon_{t+1}) \;>\; \frac{\ln t}{d}.$$

Note that $\epsilon_t - 2\epsilon_{t+1} < 1$. Using the fact that $\ln(1+x) > x/2$ for $x < 1$, we see that choosing ϵ_i such that

$$\frac{\epsilon_t - 2\epsilon_{t+1}}{2} \;=\; \frac{\ln t}{d}$$

will satisfy our requirements. Unfolding the recurrence,

$$\frac{\epsilon_1}{2^t} - \epsilon_{t+1} \;=\; \sum_{i=1}^{t} \frac{\ln i}{d2^{t-i}} \;=\; \sum_{i=1}^{t} \frac{\ln i}{d2^{t-i}} \;\le\; \sum_{i=1}^{t} \frac{\ln t}{d2^{t-i}} \;\le\; \frac{2\ln t}{d}.$$

Noting that $\epsilon_1 = 1$, we obtain that $\epsilon_{t+1} \ge 2^{-t} - 2d^{-1}\ln t$. The lower bound is the smallest value of t for which ϵ_{t+1} is negative. Let t_{max} be the largest value of t for which

$$\frac{1}{2^t} - \frac{2\ln t}{d} \;\ge\; 0.$$

This implies that $2^{t+1}\ln t \le d$ and hence

$$t_{max} \;=\; \Omega\left(\frac{\log d}{\log\log\log d}\right).$$

This gives the desired lower bound. □

Appendix. *t*-diameter-greedy algorithm. We give a few preliminary results on the t-diameter-greedy algorithm defined in section 2.

THEOREM A.1. *For $k = 3$, there is a lower bound of 3 on the performance ratio of the diameter-greedy algorithm on the line.*

Proof. We first show that diameter-greedy achieves a ratio of 3 for $k = 3$. Suppose that the optimal clustering is $[r, s], [t, u], [v, w]$ with $\max(s - r, u - t, w - v) = d$. It is sufficient to show that a merging of two out of four clusters does not create a cluster of diameter greater than $3d$. There are two cases: if $t - s, v - u > d$, then this algorithm actually produces the optimal solution; conversely, if $t - s \le d$, then either the first two out of four clusters are contained in $[r, u]$ with $u - r \le 3d$ or the last two out of four clusters are contained in $[v, w]$ with $w - v \le d$.

For the case $k = 2$, we will in fact show that diameter-greedy creates two intervals whose radius (the 2-diameter) is at most the diameter of the optimal solution. Suppose the two intervals obtained by the algorithm are $[0, a]$ and $[b, 1]$, with $a < b$. The optimum diameter is achieved when $a \le 1/2 \le b$, in which case it is $\max(a, 1 - b)$. The other case has a and b on the same side of $1/2$, say $b < 1/2$. We claim that in this case, there are no gaps greater than b between consecutive points. Consequently, the two consecutive points $x \le 1/2 \le y$ satisfy $y - x \le b$, and the optimum diameter is $d = \max(x, 1 - y)$. The interval $[0, a]$ has radius at most $a < b \le x \le d$ as needed. The interval $[b, 1]$ has radius at most $\max(y - b, 1 - y) \le \max(x, 1 - y) = d$ as needed.

It remains to verify the claim that there are no gaps greater than b. If there is such a gap, it must be inside $[b, 1]$. At some point, a merge crossing this gap was performed. That is, we had three intervals $[r, s], [t, u], [v, w]$ with $t - s > b$, and a merge producing $[r, u]$ was carried out. This can only happen if $w - u > b$. Thus, we obtain two intervals $[r, u]$ and $[v, w]$ with $b < (w - r)/2$, $v - r > b$, and $w - u > b$. We shall show that these three inequalities are preserved for the two current intervals until the end of the algorithm. However, they are false at the end for the two intervals $[0, a]$ and $[b, 1]$, a contradiction.

We show that the three inequalities are preserved. A new point z is added either between the two intervals or outside the two intervals, say after w. If z is added after w, then the resulting intervals are either $[r, u]$ and $[v, z]$, in which case the inequalities still hold, or $[r, w]$ and $[z, z]$, which can only happen if $z - w > b$ and so the inequalities still hold. If z is added between the two intervals, say $z \leq (w + r)/2$, and the two resulting intervals are $[r, z]$ and $[v, w]$, then $w - z \geq (w - r)/2 > b$, completing the proof. □

For $k = 3$, we give an example where a greedy algorithm based on diameter cannot do strictly better than 3.

THEOREM A.2. *For $k = 3$, there is a lower bound of 3 on the performance ratio of the diameter-greedy algorithm on the line.*

Proof. The adversary gives the following sequence of points: $1 + \epsilon, 2 + \epsilon, 3, 4, 6 - 2\epsilon, 4 - \epsilon, 7 - 2\epsilon$. The optimal intervals are $[1 + \epsilon, 2 + \epsilon], [3, 4], [6 - 2\epsilon, 7 - 2\epsilon]$, giving diameter 1. However, when the first four points are introduced, the interval $[2 + \epsilon, 3]$ is created by diameter-greedy; when $6 - 2\epsilon$ is added the interval $[4, 6 - 2\epsilon]$ is created; when $4 - \epsilon$ is added then the enlarged interval $[2 + \epsilon, 4 - \epsilon]$ is created; and finally when $7 - 2\epsilon$ is added either $[1 + \epsilon, 4 - \epsilon]$ or $[4, 7 - 2\epsilon]$ is created, for a factor of $[3 - 2\epsilon]$. □

The proof only gives a lower bound of 2 for the t-diameter-greedy algorithm when $t > 1$, leaving open the possibility that these algorithms may perform better than diameter-greedy. Indeed, we have the following result.

THEOREM A.3. *The 3-diameter-greedy algorithm has performance ratio 3 on the line.*

Proof. In fact, we show that it produces a clustering with 3-diameter at most the optimal diameter, and the factor of 3 follows. Assume this holds before the last two clusters are merged. Let $I_1, I_2, \ldots, I_k$ be the intervals in the optimal clustering, with maximum diameter d. Let $C_1, C_2, \ldots, C_{k+1}$ be the current clusters, each with 3-diameter at most d, of which two must be merged. If C_i starts in I_a and ends in I_b, let $x_i = b - a$; notice that $x_1 + \cdots + x_{k+1} \leq k - 1$. We assume that if C_i ends in I_b, then C_{i+1} starts in I_b; otherwise, we could replace the argument in the k intervals I_j by an argument either in the first b intervals $I_1, \ldots, I_b$ if there are at least $b + 1$ clusters C_i in this region, or in the last $k - b$ intervals $I_{b+1}, \ldots, I_k$ if there are at least $k - b + 1$ current clusters C_i in this region. Now, the bounds imply that for some i, we have $x_i + x_{i+1} < 2$. If $x_i = x_{i+1} = 0$, then the merging of C_i and C_{i+1} is contained in a single interval I_j and has a diameter at most d. If say $x_i = 0$ and $x_{i+1} = 1$, then the gap G between the two consecutive intervals I_j and I_{j+1} involved is at most d, since C_{i+1} has 3-diameter at most d, so the merger of C_i and C_{i+1} has 3-diameter at most d given by the 3-partition I_j, G, I_{j+1}. This completes the proof. □

We comment briefly on the running time of this algorithm. In the above proof, the 3-diameter of an interval may be replaced by an easily computed upper bound: at the time of creation of interval $[a, b]$, let $[x, y]$ be the gap containing $(a + b)/2$, and let the upper bound be $\max(x - a, y - x, b - y)$. Maintaining the n points sorted in a balanced tree, the running time is $O(\log n)$ for each of the n points inserted.

Acknowledgments. We thank Pankaj Agarwal and Leonidas Guibas for helpful discussions and for suggesting that we consider the dual clustering problem. We thank Bernard Chazelle for pointing out reference [5].

REFERENCES

[1] M. S. Aldenderfer and R. K. Blashfield, *Cluster Analysis*, Sage, Beverly Hills, CA, 1984.
[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, *Local search heuristic for k-median and facility location problems*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 21–29.
[3] Y. Bartal, M. Charikar, and D. Raz, *Approximating min-sum k-clustering in metric spaces*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 11–20.
[4] M. Bern and D. Eppstein, *Approximation algorithms for geometric problems*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS, Boston, MA, 1996, pp. 296–345.
[5] T. Bonnesen and W. Fenchel, *Theorie der Konvexen Körper*, Springer, Berlin, 1934; English translation: BCS Associates, Moscow, ID, 1987.
[6] P. Brucker, *On the complexity of clustering problems*, in Optimization and Operations Research, R. Henn, B. Korte, and W. Oletti, eds., Heidelberg, New York, 1977, pp. 45–54.
[7] F. Can, *Incremental clustering for dynamic information processing*, ACM Trans. Inform. Process. Systems, 11 (1993), pp. 143–164.
[8] F. Can and E. A. Ozkarahan, *A dynamic cluster maintenance system for information retrieval*, in Proceedings of the 10th Annual International ACM SIGIR Conference, 1987, pp. 123–131.
[9] F. Can and N. D. Drochak II, *Incremental clustering for dynamic document databases*, in Proceedings of the 1990 Symposium on Applied Computing, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 61–67.
[10] S. Chakrabarti, C. Phillips, A. Schulz, D. B. Shmoys, C. Stein, and J. Wein, *Improved scheduling algorithms for minsum criteria*, in Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming, Springer, Berlin, 1996, pp. 646–657.
[11] M. Charikar, S. Guha, E. Tardos, and D. Shmoys, *A constant-factor approximation algorithm for the k-median problem*, J. Comput. System Sci., 65 (2002), pp. 129-149.
[12] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan, *Algorithms for facility location problems with outliers*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 642–651.
[13] M. Charikar, L. O'Callaghan, and R. Panigrahy, *Better streaming algorithms for clustering problems*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003, pp. 30–39.
[14] M. Charikar and R. Panigrahy, *Clustering to minimize the sum of cluster diameters*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 1–10.
[15] B. B. Chaudhri, *Dynamic clustering for time incremental data*, Pattern Recognition Lett., 13 (1994), pp. 27–34.
[16] D. R. Cutting, D. R. Karger, J. O. Pederson, and J. W. Tukey, *Scatter/gather: A cluster-based approach to browsing large document collections*, in Proceedings of the 15th Annual International ACM SIGIR Conference, 1992, pp. 318–329.
[17] D. R. Cutting, D. R. Karger, and J. O. Pederson, *Constant interaction-time scatter/gather browsing of very large document collections*, in Proceedings of the 16th Annual International ACM SIGIR Conference, 1993, pp. 126–135.
[18] W. F. de la Vega, M. Karpinski, C. Kenyon, and Y. Rabani, *Approximation schemes for clustering problems*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003, pp. 50–58.
[19] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
[20] B. Everitt, *Cluster Analysis*, Heinemann Educational, London, 1974.
[21] C. Faloutsos and D. W. Oard, *A Survey of Information Retrieval and Filtering Methods*, Technical report CS-TR-3514, Department of Computer Science, University of Maryland, College Park, 1995.
[22] T. Feder and D. H. Greene, *Optimal Algorithms for Approximate Clustering*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 434–444.

[23] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto, *Optimal packing and covering in the plane are NP-complete*, Inform. Process. Lett., 12 (1981), pp. 133–137.
[24] W. Frakes and R. Baeza-Yates, eds., *Information Retrieval: Data Structures and Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
[25] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.
[26] M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, in Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 152–157.
[27] T. E. Gonzalez, *Clustering to minimize the maximum intercluster distance*, Theoret. Comput. Sci., 38 (1985), pp. 293–306.
[28] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, *Clustering data streams*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 359–366.
[29] J. A. Hartigan, *Clustering Algorithms*, Wiley, New York, 1975.
[30] D. Hochbaum, *Various notions of approximations: Good, better, best, and more*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS, Boston, MA, 1996, pp. 346–446.
[31] D. S. Hochbaum and W. Maas, *Approximation schemes for covering and packing problems in image processing and VLSI*, J. ACM, 32 (1985), pp. 130–135.
[32] D. S. Hochbaum and D. B. Shmoys, *A best possible heuristic for the k-center problem*, Math. Oper. Res., 10 (1985), pp. 180–184.
[33] D. S. Hochbaum and D. B. Shmoys, *A unified approach to approximation algorithms for bottleneck problems*, J. ACM, 33 (1986), pp. 533–550.
[34] S. Irani and A. Karlin, *Online computation*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS, Boston, MA, 1996, pp. 521–564.
[35] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[36] K. Jain and V. Vazirani, *Approximation algorithms for metric facility location and k-Median problems using the primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.
[37] N. Jardine and C. J. van Rijsbergen, *The use of hierarchical clustering in information retrieval*, Inform. Storage and Retrieval, 7 (1971), pp. 217–240.
[38] O. Kariv and S. L. Hakimi, *An algorithmic approach to network location problems.* I. *The p-centers*, SIAM J. Appl. Math., 37 (1979), pp. 513–538.
[39] N. Megiddo and K. J. Supowit, *On the complexity of some common geometric location problems*, SIAM J. Comput., 13 (1984), pp. 182–196.
[40] S. G. Mentzer, *Lower Bounds on Metric k-Center Problems*, manuscript, 1988.
[41] R. Motwani, S. Phillips, and E. Torng, *Nonclairvoyant scheduling*, Theoret. Comput. Sci., 130 (1994), pp. 17–47.
[42] E. Omiecinski and P. Scheuermann, *A global approach to record clustering and file organization*, in Proceedings of the 3rd BCS-ACM Symposium on Research and Development in Information Retrieval, 1984, pp. 201–219.
[43] J. Pach and P. K. Agarwal, *Combinatorial Geometry*, Wiley, New York, 1995.
[44] E. Rasmussen, *Clustering algorithms*, in Information Retrieval: Data Structures and Algorithms, W. Frakes and R. Baeza-Yates, eds., Prentice-Hall, Englewood Cliffs, NJ, 1992, pp. 419–442.
[45] C. J. van Rijsbergen, *Information Retrieval*, Butterworths, London, 1979.
[46] C. Rogers, *A note on coverings*, Mathematika, 4 (1957), pp. 1–6.
[47] G. Salton, *Automatic Text Processing*, Addison-Wesley, Reading, MA, 1989.
[48] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983.
[49] P. Willett, *Recent trends in hierarchical document clustering: A critical review*, Inform. Process. Management, 24 (1988), pp. 577–597.
[50] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, Van Nostrand Reinhold, New York, 1994.

SIAM J. COMPUT.
Vol. 33, No. 6, pp. 1441–1483

TIGHT BOUNDS FOR TESTING BIPARTITENESS IN GENERAL GRAPHS*

TALI KAUFMAN†, MICHAEL KRIVELEVICH‡, AND DANA RON§

Abstract. In this paper we consider the problem of testing bipartiteness of general graphs. The problem has previously been studied in two models, one most suitable for dense graphs and one most suitable for bounded-degree graphs. Roughly speaking, dense graphs can be tested for bipartiteness with constant complexity, while the complexity of testing bounded-degree graphs is $\tilde{\Theta}(\sqrt{n})$, where n is the number of vertices in the graph (and $\tilde{\Theta}(f(n))$ means $\Theta(f(n) \cdot \text{polylog}(f(n)))$). Thus there is a large gap between the complexity of testing in the two cases.

In this work we bridge the gap described above. In particular, we study the problem of testing bipartiteness in a model that is suitable for all densities. We present an algorithm whose complexity is $\tilde{O}(\min(\sqrt{n}, n^2/m))$, where m is the number of edges in the graph, and we match it with an almost tight lower bound.

Key words. property testing, bipartiteness, randomized algorithms

AMS subject classifications. 68Q25, 68R10

DOI. 10.1137/S0097539703436424

1. Introduction.

Property testing algorithms [21, 12] are algorithms that perform *approximate decisions*. Namely, for a predetermined property P they should decide whether a given object O has property P or is *far* from having property P. In order to perform this approximate decision they are given *query access* to the object O. Property testing problems are hence defined by the type of objects in question, the property tested, the type of queries allowed, and the notion of distance to having a property. Much of the focus of property testing has been on testing properties of graphs. In this context several models have been considered. In all models, for a fixed graph property P, the algorithm is required to accept graphs that have P and to reject graphs that are ϵ-far from having P for a given distance parameter ϵ. In all cases the algorithm is allowed a constant probability of failure. The models differ in the type of queries they allow and in the notion of distance they use (which underlies the definition of being ϵ-far from having the property). The complexity of the algorithm is measured by the number of queries that the algorithm performs.

1.1. Models for testing graph properties.

The first model, introduced in [12], is the adjacency-matrix model. In this model the algorithm may perform queries of the following form: "Is there an edge between vertices u and v in the graph?" That is, the algorithm may probe the adjacency matrix representing the graph. We refer to such queries as *vertex-pair* queries. The notion of distance is also linked to this representation: a graph is said to be ϵ-far from having property P if more

*Received by the editors October 17, 2003; accepted for publication (in revised form) April 9, 2004; published electronically September 2, 2004.
http://www.siam.org/journals/sicomp/33-6/43642.html

†Department of Computer Science, Tel Aviv University, Tel Aviv, Israel (kaufmant@post.tau.ac.il). This work is part of this author's Ph.D. thesis prepared at Tel Aviv University under the supervision of Prof. Noga Alon and Prof. Michael Krivelevich.

‡Department of Mathematics, Tel Aviv University, Tel Aviv, Israel (krivelev@post.tau.ac.il). This author's research was supported in part by a USA Israel BSF grant and by a grant from the Israel Science Foundation.

§Department of EE – Systems, Tel Aviv University, Tel Aviv, Israel (danar@eng.tau.ac.il). This author's research was supported by the Israel Science Foundation (grant 32/00-1).

than ϵn^2 edge modifications should be performed on the graph so that it obtains the property, where n is the number of vertices in the graph. In other words, ϵ measures the fraction of entries in the adjacency matrix of the graph that should be modified. This model is most suitable for *dense* graphs in which the number of edges, denoted m, is $\Theta(n^2)$. This model was studied in [12, 3, 2, 1, 4, 16, 8].

The second model, introduced in [13], is the (bounded-degree) incidence-lists model. In this model, the algorithm may perform queries of the following form: "Who is the ith neighbor of vertex v in the graph?" That is, the algorithm may probe the incidence lists of the vertices in the graph, where it is assumed that all vertices have degree at most d for some fixed degree-bound d. We refer to these queries as *neighbor* queries. Here too the notion of distance is linked to the representation: A graph is said to be ϵ-far from having property P if more than ϵdn edge modifications should be performed on the graph so that it obtains the property. In this case ϵ measures the fraction of entries in the incidence-lists representation (among all dn entries) that should be modified. This model is most suitable for graphs with $m = \Theta(dn)$ edges, that is, whose maximum degree is of the same order as the average degree. In particular, this is true for *sparse* graphs that have *constant degree*. This model was studied in [14, 13, 7].

In [20] it was suggested to decouple the issues of representation and type of queries allowed from the definition of distance to having a property. Specifically, it was suggested to measure the distance simply with respect to the number of edges, denoted m, in the graph. Namely, a graph is said to be ϵ-far from having a property if more than ϵm edge modifications should be performed so that it obtains the property. In [20] the algorithm was allowed the same type of queries as in the bounded-degree incidence-lists model, but no fixed upper-bound was assumed on the degrees and the algorithm could query the degree of any vertex. The main advantage of this model over the bounded-degree incidence-lists model is that it is suitable for graphs whose degrees may vary significantly. To illustrate this, consider a (sparse) graph having $m = O(n)$ edges, where the maximum degree d of vertices in the graph is $\Omega(n)$. Suppose we want to determine whether the graph is bipartite or ϵ-far from being bipartite for some constant ϵ. If we worked in the bounded-degree incidence-lists model then we could trivially accept all graphs, since for $d = \Omega(n)$ and constant ϵ every graph having $O(n) = o(dn)$ edges is ϵ-close to being bipartite. However, this is no longer true when distance is measured with respect to the actual number of edges m.

The model studied in this paper. In this work we are interested in a model that may be useful for testing all types of graphs: dense, sparse, and graphs that lie in-between the two extremes. As is discussed in more detail in the next subsection, the two extremes sometimes exhibit very different behavior in terms of the complexity of testing the same property. We are interested in understanding the transformation from testing sparse (and, in particular, bounded-degree) graphs to testing dense graphs.

Recall that a model for testing graph properties is defined by the distance measure used and by the queries allowed. The model of [20] is indeed suitable for all graphs in terms of the distance measure used, since distance is measured with respect to the actual number of edges m in the graph. Thus this notion of distance adapts itself to the density of the graph, and we shall use it in our work.

The focus in [20] was on testing properties that are of interest in sparse (but not necessarily bounded-degree) graphs, and hence they allowed only neighbor queries. However, consider the case in which the graph is not sparse (but not necessarily dense).

In particular, suppose that the graph has $\omega(n^{1.5})$ edges and that we are seeking an algorithm that performs $o(\sqrt{n})$ queries. While in the case of sparse graphs, there is no use in asking vertex-pair queries (i.e., is there an edge between a particular pair of vertices), such queries may become helpful when the number of edges is sufficiently large. Hence, we allow our algorithms to perform both neighbor queries and vertex-pair queries.

1.2. Testing bipartiteness. One of the properties that has received quite a bit of attention in the context of property testing is *bipartiteness*. Recall that a graph is bipartite if it is possible to partition its vertices into two parts such that there are no edges with both endpoints in the same part. This property was first studied in [12] where it was shown that bipartiteness can be tested in the adjacency-matrix model by a simple algorithm using $\tilde{O}(1/\epsilon^3)$ queries. This was improved in [3] to $\tilde{O}(1/\epsilon^2)$ queries. The best lower bound known in this model is $\tilde{\Omega}(1/\epsilon^{1.5})$, due to [8]. Thus the complexity of this problem in the adjacency-matrix model is independent of the number of vertices n and is polynomial in $1/\epsilon$. It is interesting to note that testing bipartiteness is considered implicitly already in [9]. The result in [9] can be used to obtain a testing algorithm in the adjacency-matrix model whose query complexity does not depend on the size of the graph but whose dependence on ϵ is a tower of height polynomial in $1/\epsilon$.

The complexity of testing bipartiteness is significantly different when considering the bounded-degree incidence-lists model. In [14] a lower bound of $\Omega(\sqrt{n})$ was established in this model for constant ϵ and constant d (where d is the degree bound). An almost matching upper bound of $\tilde{O}(\sqrt{n} \cdot \mathrm{poly}(1/\epsilon))$ was shown in [13]. Thus, in the case of bipartiteness there is a large gap between the results that can be obtained for dense graphs and for constant-degree graphs.

Here we venture into the land of graphs that are neither necessarily sparse nor necessarily dense and study the complexity of testing bipartiteness. As we discuss briefly in subsection 1.5, other graph properties exhibit similar (and sometimes even larger) gaps, and hence we believe that understanding the transformation from sparse to dense graphs is of general interest.

1.3. Our results. In this work we present two complementary results for n-vertex graphs having m edges:

- We describe and analyze an algorithm for testing bipartiteness in general graphs whose query complexity and running time are $O(\min(\sqrt{n}, n^2/m) \cdot \mathrm{poly}(\log n/\epsilon))$. The algorithm has a one-sided error (i.e., it always accepts bipartite graphs). Furthermore, whenever it rejects a graph it provides *evidence* that the graph is not bipartite in the form of an odd cycle[1] of length $\mathrm{poly}(\log n/\epsilon)$.
- We present an almost matching lower bound of $\Omega(\min(\sqrt{n}, n^2/m))$ (for a constant ϵ). This bound holds for all testing algorithms (that is, for those that are allowed a two-sided error and are adaptive). Furthermore, the bound holds for regular graphs.

As seen from the above expressions, as long as $m = O(n^{1.5})$, that is, the average degree is $O(\sqrt{n})$, the complexity of testing (in terms of the dependence on n) is $\tilde{\Theta}(\sqrt{n})$. Once the number of edges goes above $n^{1.5}$, we start seeing a decrease in the query complexity, which in this case is at most $O((n^2/m) \cdot \mathrm{poly}(\log n/\epsilon))$. In terms of our algorithm, this is exactly the point where our algorithm starts exploiting its

[1] We use the term "odd cycle" as a shorthand for "odd-length cycle."

access to vertex-pair queries. Our lower bound shows that this behavior of the query complexity is not only an artifact of our algorithm but is inherent in the problem.

Notes.

1. Observe that even if the graph is sparse, we obtain a new result that does not follow from [13]. Namely, we have an algorithm with complexity $\tilde{O}(\sqrt{n} \cdot \text{poly}(1/\epsilon))$ for sparse graphs with varying degrees.
2. We note that the algorithm does not actually require to be given the number of edges, m, in the graph but can instead compute an estimate of this number. Such an estimate can be obtained without increasing the query complexity of the algorithm [11, 15], and we discuss this issue shortly in section 4.
3. We assume that $m = \Omega(n)$. Since our distance measure is with respect to the number of edges in the graph, without such an assumption it would be impossible to distinguish between the following two (families of) graphs: a graph that consists of many isolated edges and a single very small subgraph that is far from bipartite (e.g., a clique), and a graph that consists of many isolated vertices and a very small bipartite subgraph. We discuss this issue briefly in section 4 as well.

1.4. Our techniques. We present our algorithm in two stages. First we describe an algorithm that works for almost-regular graphs, that is, graphs in which the maximum degree is of the same order as the average degree. The algorithm and its analysis closely follow the algorithm and analysis in [13]. Indeed, as long as the degree d of the graph is at most $\sqrt{n}$, we execute the algorithm described in [13]. The place where we depart from [13] is in the usage of vertex-pair queries once $d > \sqrt{n}$. We refer to our first algorithm as Test-Bipartite-Reg.

In the second stage we show how to reduce the problem of testing bipartiteness of general graphs to testing bipartiteness of almost-regular graphs. Namely, we show how, for every given graph G, it is possible to define a graph G' such that (1) G' has roughly the same number of vertices and edges as G, and its maximum degree is of the same order as its average degree (which is roughly the same as the average degree in G); (2) if G is bipartite, then so is G', and if G is far from being bipartite, then so is G'. We then show how to emulate the execution of the algorithm Test-Bipartite-Reg on G' given query access to G so that we may accept G if it accepts G' and reject G if it rejects G'.

In the course of this emulation we are confronted with the following interesting problem: We would like to sample vertices in G according to their degrees (which aids us in sampling vertices uniformly in G', a basic operation that is required by Test-Bipartite-Reg). The former is equivalent to sampling *edges* uniformly in G. In order not to harm the performance of our testing algorithm, we are required to perform this task using $\tilde{O}(\min(\sqrt{n}, n^2/m))$ queries. If m is sufficiently large (once again, if $m \geq n^{1.5}$), this can be performed without increasing the complexity of our algorithm simply by sampling sufficiently many pairs of vertices in G. However, we do not know how to perform this task exactly (in an efficient manner) when the number of edges is significantly smaller than $n^{1.5}$. Nonetheless, we provide a sampling procedure that selects edges according to a distribution that approximates the desired uniform distribution on edges and is sufficient for our purposes. The approximation is such that for all but a small fraction of the m edges, the probability of selecting an edge is $\Omega(1/m)$. This procedure may be of independent interest.

We also conjecture that variants of our construction of G' (and, in particular, a simple probabilistic construction we suggest) may be useful in transforming other

results that hold for graphs whose maximum degree is similar to their average degree into results that hold for graphs with varying degrees.

We establish our lower bound by describing, for every pair n, d (n even, $d \geq 64$), two distributions over d-regular graphs. In one distribution all graphs are bipartite by construction. For the other distribution we prove that almost all graphs are far from being bipartite. We then show that every testing algorithm that can distinguish between a graph chosen randomly from the first distribution (which it should accept with probability at least 2/3) and a graph chosen randomly from the second distribution (which it should reject with probability at least 2/3) must perform $\Omega(\min(\sqrt{n}, n/d)) = \Omega(\min(\sqrt{n}, n^2/m))$ queries.

Our lower-bound proof implies the necessity of both neighbor queries and vertex-pair queries in obtaining an upper bound whose dependence on n and m is $\tilde{O}(\min(\sqrt{n}, n^2/m))$. Specifically, if only neighbor queries are allowed, then our analysis implies a lower bound of $\Omega(\sqrt{n})$ for every m, which is higher than $\tilde{O}(n^2/m)$ when $m = \omega(n^{1.5})$. If only vertex-pair queries are allowed, then our analysis implies a lower bound of $\Omega(n^2/m)$, which is above the upper bound of $\tilde{O}(\sqrt{n})$ when $m = o(n^{1.5})$.

1.5. Further research. As noted previously, there are other problems that exhibit a significant gap between the query complexity of testing dense graphs (in the adjacency-matrix model) and the complexity of testing sparse bounded-degree graphs (in the bounded-degree incidence-lists model). In particular, this is true for testing k-colorability. It is possible to test dense graphs for k-colorability using poly(k/ϵ) queries [12, 3, 10], while testing sparse graphs requires $\Omega(n)$ queries [7]. We stress that these bounds are for query complexity, where we put time complexity aside. We would like to understand this transformation from essentially constant complexity (for constant k and ϵ) to linear complexity, and we would like to know whether any intermediate results can be obtained for graphs that are neither sparse nor dense. Other problems of interest are testing whether a graph has a relatively large clique [12], testing acyclicity of directed graphs [6], and testing that a graph does not contain a certain subgraph [1, 4].

1.6. Organization of the paper. In section 2 we give some basic definitions and notation. In sections 3 and 4 we describe and analyze our testing algorithms. In section 5 we present our lower bound.

2. Preliminaries. Let $G = (V, E)$ be an undirected graph with n vertices labeled $1, \ldots, n$, and let $m = m(G) = |E(G)|$ be the total number of edges in G. Unless stated otherwise, we assume that G contains no multiple edges. For each vertex $v \in V$ let $\Gamma(v)$ denote its set of neighbors, and let $\deg(v) = |\Gamma(v)|$ denote its degree. The edges incident to v are labeled from 1 to $\deg(v)$. We make no assumption on the corresponding order of the neighbors of a vertex. Note that each edge has two possibly different labels, one with respect to each of its endpoints. We hence view edges as quadruples. That is, if there is an edge between v and u and it is the ith edge incident to v and the jth edge incident to u, then this edge is denoted by (u, v, i, j). When we want to distinguish between the quadruple (u, v, i, j) and the pair (u, v), then we refer to the latter as an *edge-pair*. We let $d_{\max} = d_{\max}(G)$ denote the maximum degree in the graph G and $d_{\text{avg}} = d_{\text{avg}}(G)$ denote the average degree in the graph (that is, $d_{\text{avg}}(G) = 2m(G)/n$).

Distance to having a property. Consider a fixed graph property $\mathcal{P}$. For a given graph G, let $e_{\mathcal{P}}(G)$ be the minimum number of edges that should be added to G or removed from G so that it obtains property $\mathcal{P}$. The distance of G to having property $\mathcal{P}$

is defined as $e_{\mathcal{P}}(G)/m(G)$. In particular, we say that graph G is ϵ-*far* from having the property $\mathcal{P}$ for a given distance parameter $0 \leq \epsilon < 1$ if $e_{\mathcal{P}}(G) > \epsilon \cdot m(G)$. Otherwise, it is ϵ-*close* to having property $\mathcal{P}$. In some cases we may define the distance to having a property with respect to an upper bound $m_{\max} \geq m(G)$ on the number of edges in the graph (that is, the distance to having property $\mathcal{P}$ is defined as $e_{\mathcal{P}}(G)/m_{\max}$). For example, if the graph is dense so that $m(G) = \Omega(n^2)$, then we set $m_{\max} = n^2$, and alternatively, if the graph has some bounded degree d, then we set $m_{\max} = d \cdot n$. (In the latter case we could set $m_{\max} = (d \cdot n)/2$, but for simplicity we set the slightly higher upper bound.) If $e_{\mathcal{P}}(G)/m_{\max} > \epsilon$, then we shall say that the graph is ϵ-far from having property $\mathcal{P}$ *with respect to* $m_{\max}$.

Testing algorithms. A testing algorithm for a graph property $\mathcal{P}$ is required to accept with probability at least 2/3 every graph that has property $\mathcal{P}$ and to reject with probability at least 2/3 every graph that is ϵ-far from having property $\mathcal{P}$, where ϵ is a given distance parameter. If the algorithm always accepts graphs that have the property, then it is a *one-sided error* algorithm. The testing algorithm is given the number of vertices in the graph, the number of edges in the graph, or an upper bound on this number,[2] and it is provided with *query access* to the graph. Specifically, we allow the algorithm the following types of queries.

- The first type of queries is *degree* queries. That is, for any vertex u of its choice, the algorithm can obtain $\deg(u)$. We assume that a degree query has cost one.
- The second type of queries is *neighbor* queries. Namely, for every vertex u and index $1 \leq i \leq \deg(u)$, the algorithm may obtain the ith neighbor of vertex u.
- The third type of queries is *vertex-pair* queries. Namely, for any pair of vertices (u, v), the algorithm can query whether there is an edge between u and v in G.

Note that degree queries can be easily implemented using neighbor queries with cost $O(\log d_{\max}) = O(\log n)$.

Bipartiteness. In this work we focus on the property of *bipartiteness*. Let (V_1, V_2) be a partition of V. We say that an edge $(u, v) \in E$ is a *violating* edge with respect to (V_1, V_2) if u and v belong to the same subset V_b (for some $b \in \{1, 2\}$). A graph is *bipartite* if there exists a partition of its vertices with respect to which there are no violating edges. By definition, a graph is ϵ-far from being bipartite if for every partition of its vertices, the number of violating edges with respect to the partition is greater than $\epsilon \cdot m$. Recall that a graph is bipartite if and only if it contains no odd cycles.

3. The algorithm for the almost-regular case. In this section we describe an algorithm that accepts every bipartite graph and that rejects with probability at least 2/3 every graph that is ϵ-far from being bipartite with respect to an upper bound $m_{\max} = d_{\max} n$ on the number of edges. Namely, this algorithm rejects (with probability at least 2/3) graphs for which the number of edges that need to be removed so that they become bipartite is greater than $\epsilon \cdot m_{\max} = \epsilon \cdot d_{\max} n$. The query complexity and running time of this algorithm are $O(\min(\sqrt{n}, n/d_{\max}) \cdot \mathrm{poly}(\log n/\epsilon))$.

In the case where the graph is almost regular, that is, the maximum degree of the graph $d_{\max}$ is of the same order as the average degree, d_{avg}, then we essentially obtain

[2] As noted in the introduction, we can remove this assumption and have the algorithm compute an estimate of the number of edges.

a tester as desired (since in such a case $\epsilon d_{\max} n = O(\epsilon m)$). However, in general, $d_{\max}$ may be much larger than d_{avg} (for example, it is possible that $d_{\max} = \Theta(n)$ while $d_{\text{avg}} = \Theta(1)$). To deal with the general case we show in the next section (section 4) how to reduce the problem in the general case to the special case of $d_{\max} = O(d_{\text{avg}})$.

Test-Bipartite-Reg($n, d_{\max}, \epsilon$)

- Repeat $T = \Theta(\frac{1}{\epsilon})$ times:
  1. Uniformly select a vertex s in V.
  2. If Odd-Cycle(s) returns found, then output reject.
- In case no call to Odd-Cycle returned found, then output accept.

Odd-Cycle(s)

1. If $d = d_{\max} \leq \sqrt{n}$, then let $K \stackrel{\text{def}}{=} \Theta\left(\frac{\sqrt{n} \cdot \log^{1/2}(n/\epsilon)}{\epsilon^3}\right)$ and $L \stackrel{\text{def}}{=} \Theta\left(\frac{\log^3(n/\epsilon)}{\epsilon^5}\right)$. Otherwise ($d > \sqrt{n}$), let $K \stackrel{\text{def}}{=} \Theta\left(\frac{\sqrt{n/d} \cdot \log^{1/2}(n/\epsilon)}{\epsilon^8}\right)$, and $L \stackrel{\text{def}}{=} \Theta\left(\frac{\log^6(n/\epsilon)}{\epsilon^8}\right)$.
2. Perform K random walks starting from s, each of length L.
3. Let A_0 (A_1) be the set of vertices that appear at the ends of the walks performed in the previous step whose paths are of even (odd) length.
4. If $d \leq \sqrt{n}$, then check whether $A_0 \cap A_1 \neq \emptyset$. If the intersection is nonempty, then return found; otherwise return not-found.
5. Else ($d > \sqrt{n}$), perform vertex-pair queries between every pair of vertices $u, v \in A_0$ ($u, v \in A_1$). If an edge is detected, then return found; otherwise return not-found.

FIG. 1. *Algorithm Test-Bipartite-Reg for testing bipartiteness with respect to the upper bound $m_{\max} = d_{\max} \cdot n$ on the number of edges, and the procedure Odd-Cycle for detecting odd cycles in the graph G.*

A high level description of the algorithm. Throughout this section let $d = d_{\max}$. Our algorithm, whose pseudocode appears in Figure 1, builds on the testing algorithm for bipartiteness described in [13]. The query complexity of that algorithm is $O(\sqrt{n} \cdot \text{poly}(\log n/\epsilon))$, and it works with respect to $m_{\max} = dn$ as well. In fact, as long as $d \leq \sqrt{n}$, our algorithm is the same as the algorithm in [13].

In particular, as in [13], our algorithm selects $\Theta(1/\epsilon)$ *starting vertices* and from each it performs several random walks (using neighbor queries), each walk of length $\text{poly}(\log n/\epsilon)$. The exact form of these random walks is described momentarily. If $d \leq \sqrt{n}$, then the number of these random walks from each starting vertex s is $O(\sqrt{n} \cdot \text{poly}(\log n/\epsilon))$, and the algorithm simply checks whether an odd cycle was detected in the course of these random walks. Specifically, the algorithm checks whether there exists some vertex v that is reached at the end of two different walks from s, where one walk corresponds to a path in the graph with even length, and one walk corresponds to a path with odd length. The existence of such a vertex v implies an odd cycle that contains s and v, and the algorithm rejects the graph in such a case.

If $d > \sqrt{n}$, then there are two important modifications as compared to the case $d \leq \sqrt{n}$ (which, as noted above, follows [13]). These modifications reduce the number of queries performed as the degree increases.

1. The number of random walks performed from each starting vertex is reduced to $O(\sqrt{n/d} \cdot \text{poly}(\log n/\epsilon))$ (as compared to $O(\sqrt{n} \cdot \text{poly}(\log n/\epsilon))$ walks for the case $d \leq \sqrt{n}$).

2. The algorithm still considers the vertices reached at the end of these walks, but now it performs an additional step. It partitions these vertices into two subsets, denoted A_0 and A_1, according to the parity of the lengths of the paths corresponding to the walks. The algorithm then performs vertex-pair queries on each pair of vertices that belong to the same subset. If any edge (u, v) is detected for $u, v \in A_0$ or $u, v \in A_1$, then the algorithm has evidence of an odd cycle that included s (the starting vertex), u, and v, and it rejects. The total number of vertex-pair queries is $O((n/d) \cdot \text{poly}(\log n/\epsilon))$.

On a very intuitive level, if a graph is far from being bipartite, then many edges (and vertices) belong to many odd cycles. The difference between the two cases described above is that if $d \leq \sqrt{n}$, then the algorithm tries to reach the same vertex twice, once via an even-length path and once via an odd-length path. To this end it performs about $\sqrt{n}$ random walks (so as to "hit" the same vertex twice). In the case $d > \sqrt{n}$, the algorithm performs far fewer walks, and so we cannot expect to reach the same vertex twice. However, since the edge-density is higher, we do expect to have an edge in the subgraph induced by the ending points of the walk. In particular, as our analysis shows, we expect to see such an edge between vertices that are reached via paths whose lengths have the same parity.

Random walks and paths in the graph. The random walks performed are defined as follows: At each step, if the degree of the current vertex v is $d' \leq d$, then the walk *remains* at v with probability $1 - \frac{d'}{2d} \geq \frac{1}{2}$, and for each $u \in \Gamma(v)$, the walk *traverses* to u with probability $\frac{1}{2d}$. The important property of the random walk is that the stationary distribution it induces over the vertices is uniform.

To every walk (or, more generally, to any sequence of steps), there corresponds a *path* in the graph. The path is determined by those steps in which an edge is traversed (while ignoring all steps in which the walk stays at the same vertex). Such a path is not necessarily simple but does not contain self-loops. Note that when referring to the length of a walk, we mean the total number of steps taken, including steps in which the walk remains at the current vertex, while the length of the corresponding path does not include these steps.

THEOREM 1. *The algorithm Test-Bipartite-Reg accepts every graph that is bipartite and rejects with probability at least* 2/3 *every graph that is* ϵ*-far from being bipartite with respect to* $m_{\max} = d_{\max} n$*. Furthermore, whenever the algorithm rejects a graph, it outputs a* certificate *to the nonbipartiteness of the graph in the form of an odd cycle of length* $\text{poly}(\log n/\epsilon)$*. The query complexity and running time of the algorithm are* $O\left(\min(\sqrt{n}, n/d_{\max}) \cdot \text{poly}(\log n/\epsilon)\right)$.

Note that the algorithm can work when G contains self-loops and multiple edges. The latter will be of importance in the next section.

As a direct corollary of Theorem 1 (using $m(G) = (n d_{\text{avg}}(G))/2$), we get the following.

COROLLARY 2. *For a given graph* G*, let* $\gamma(G) \stackrel{\text{def}}{=} d_{\max}(G)/d_{\text{avg}}(G)$*. Then* Test-Bipartite-Reg$(n, d_{\max}(G), \epsilon/(2\gamma(G)))$ *accepts every graph that is bipartite and rejects with probability at least* 2/3 *every graph that is* ϵ*-far from being bipartite (with respect to* $m(G)$*).*

The corollary below will become useful in the next section.

COROLLARY 3. *If* G *is* ϵ*-far from being bipartite with respect to* $m_{\max} = d_{\max} n$*, then* $\Omega(\epsilon)$*-fraction of its vertices* s *are such that* Odd-Cycle(s) *returns* found *with probability at least* $\frac{2}{3}$.

The completeness part of Theorem 1 (i.e., showing that the algorithm accepts bi-

partite graphs) is straightforward. We focus on proving the soundness of the algorithm (i.e., that graphs that are ϵ-far from being bipartite are rejected with probability $\frac{2}{3}$). What we eventually show (in subsection 3.6) is the contrapositive statement, namely, that if the test accepts G with probability greater than $\frac{1}{3}$, then there exists an ϵ-good partition of G.

Our analysis follows the analysis presented in [13] quite closely. In particular, whenever possible we refer the reader to proofs given in [13]. Here we present what is necessary to establish the correctness of our algorithm and in particular those proofs in which we diverge from [13]. Since the algorithm for the case $d_{\max} \leq \sqrt{n}$ is fully analyzed in [13], from this point on we assume $d_{\max} > \sqrt{n}$ and analyze the algorithm for this case.

3.1. Gaining intuition: The rapidly mixing case. To gain some intuition, consider first the following "ideal" case: From each starting vertex s in G, and for every $v \in V$, the probability that a random walk of length $L = \text{poly}((\log n)/\epsilon)$ ends at v is at least $\frac{1}{2n}$ and at most $\frac{2}{n}$, i.e., approximately the probability assigned by the stationary distribution. (Note that this ideal case occurs when G is an expander.) Let us fix a particular starting vertex s. For each vertex v, let p_v^0 be the probability that a random walk (of length L) starting from s ends at v and corresponds to an even-length path. Define p_v^1 analogously for odd paths. Then, by our assumption on G, for every v, $p_v^0 + p_v^1 \geq \frac{1}{2n}$. We consider two cases regarding the sum $\sigma(G) \stackrel{\text{def}}{=} \sum_{\substack{v,u \in V \\ (v,u) \in E}} (p_v^0 p_u^0 + p_v^1 p_u^1)$.

In case $\sigma(G)$ is (relatively) "small," we show that there exists a partition (V_0, V_1) of V that is ϵ-good, and so G is ϵ-close to being bipartite. Otherwise (i.e., when the sum is not "small"), we show that the rejection probability is bounded away from zero. This implies that in case G is accepted with probability at least $\frac{1}{3}$, then G is ϵ-close to being bipartite.

Consider first the case in which $\sigma(G) < c \cdot \frac{\epsilon d}{n}$ for some suitable constant $c < 1$. Let the partition (V_0, V_1) be defined as follows: $V_0 = \{v : p_v^0 \geq p_v^1\}$ and $V_1 = \{v : p_v^1 > p_v^0\}$. Consider a particular vertex $v \in V_0$. By definition of V_0 and our rapid-mixing assumption, $p_v^0 \geq \frac{1}{4n}$.

$$\begin{aligned}
\sigma(G) &= \sum_{\substack{v,u \in V \\ (v,u) \in E}} (p_v^0 p_u^0 + p_v^1 p_u^1) \\
&\geq \sum_{\substack{v,u \in V_0 \\ (v,u) \in E}} p_v^0 p_u^0 \quad + \quad \sum_{\substack{v,u \in V_1 \\ (v,u) \in E}} p_v^1 p_u^1 \\
&\geq \sum_{\substack{v,u \in V_0 \\ (v,u) \in E}} \frac{1}{16n^2} + \sum_{\substack{v,u \in V_1 \\ (v,u) \in E}} \frac{1}{16n^2} \\
&\geq \frac{1}{16n^2} \cdot (\text{the number of violating edges with respect to } (V_0, V_1)). \qquad (1)
\end{aligned}$$

Thus, if there are more than ϵdn violating edges with respect to (V_0, V_1), then $\sigma(G) > \frac{1}{16} \cdot \frac{\epsilon d}{n}$, which contradicts our case hypothesis concerning $\sigma(G)$ assuming $c \leq 1/16$.

We now turn to the second case, $\sigma(G) \geq c \cdot \frac{\epsilon d}{n}$. For every fixed pair $i, j \in \{1, \ldots, K\}$ (recall that $K = \Theta(\sqrt{n/d} \cdot \text{poly}(\log n/\epsilon))$ is the number of walks taken from s), consider the 0/1 random variable $\eta_{i,j}$ that is 1 if and only if both the ith

and the jth walks have path length with the same parity and if the endpoints of the paths are vertices u, v such that $(u, v) \in E$. Then for every pair i, j,

$$\text{Exp}[\eta_{i,j}] \;=\; \sum_{u,v \in V,\,(u,v)\in E} (p_v^0 p_u^0 + p_v^1 p_u^1) \;=\; \sigma(G). \tag{2}$$

Since there are $K^2 = \Theta(n/d \cdot \text{poly}(\log n/\epsilon))$ such pairs i, j, the expected value of the sum over all $\eta_{i,j}$'s is greater than some constant $c' > c$. These random variables are not pairwise independent; nonetheless we can obtain a constant bound on the probability that the sum is 0 using Chebyshev's inequality (cf. [5, sec. 4.3]).

Unfortunately, we may not assume in general that for every (or even some) starting vertex, all (or even almost all) vertices are reached with probability $\Theta(1/n)$. However, roughly speaking, we are able to show that every graph can be partitioned into parts such that within each part we can perform an analysis that builds on the ideas presented above. Furthermore, the different parts are separated by small cuts so that if each part is close to being bipartite, then so is the whole graph. An important component in the analysis is the definition of the Markov chain $\text{M}_{\ell_1}^{\ell_2}(H)$, and we turn to this definition in the next subsection.

3.2. The Markov chain $\mathbf{M}_{\boldsymbol{\ell_1}}^{\boldsymbol{\ell_2}}(\boldsymbol{H})$. Let H be an induced subgraph of G. For any given pair of lengths, ℓ_1 and ℓ_2, we define a Markov chain $\text{M}_{\ell_1}^{\ell_2}(H)$. Roughly speaking, $\text{M}_{\ell_1}^{\ell_2}(H)$ captures random walks of length at most $\ell_1 \cdot \ell_2$ in G that do not exit H for (sub)walks of length ℓ_2 or more. The states of the chain consist of the vertices of H and some additional auxiliary states. For vertices that do not have neighbors outside of H, the transition probabilities in $\text{M}_{\ell_1}^{\ell_2}(H)$ are exactly as in walks on G. However, for vertices v that have neighbors outside of H there are two modifications: (1) For each vertex u, the transition probability from v to u, denoted $q_{v,u}$, is the probability of a walk (in G) starting from v and ending at u after less than ℓ_2 steps (without passing through any other vertex in H). Thus, walks of length less than ℓ_2 out of H (and, in particular, the walk $v - u$ in case $(v, u) \in E$) are contracted into single transitions. Note that for every u and v in H we have $q_{u,v} = q_{v,u}$. (2) There is an auxiliary path of length ℓ_1 emitting from v. The transition probability from v to the first auxiliary vertex on the path equals the probability that a walk starting from v exits H and does not return in less than ℓ_2 steps. From the last vertex on the auxiliary path there are transitions to vertices in H with the corresponding conditional probabilities of reaching them after such a walk.

A more formal definition of $\text{M}_{\ell_1}^{\ell_2}(H)$ appears in the appendix, together with an illustration (see Figure 8). The following definition and lemma will be instrumental in our analysis.

DEFINITION 3.1. *We say that a vertex s is* useful *with respect to $\text{M}_{\ell_1}^{\ell_2}(H)$ if the probability that a walk in $\text{M}_{\ell_1}^{\ell_2}(H)$ starting from s enters an auxiliary path after at most ℓ_1 steps is at most $\frac{2\ell_1}{\ell_2} \cdot \frac{n}{|H|}$.*

LEMMA 1. *Let H be a subgraph of G, and let ℓ_1 and ℓ_2 be integers. Then at least half of the vertices s in H are useful with respect to $\text{M}_{\ell_1}^{\ell_2}(H)$.*

The proof of the lemma appears in [13].

3.3. Useful vertices and small cuts. The following lemma can be viewed as presenting a "contrapositive statement" of the work of Mihail [19]. While Mihail showed that high expansion leads to fast convergence of random walks to the stationary distribution, the lemma below shows that too slow of a convergence implies

small cuts that have certain additional properties. In particular, the vertices on one side of the cut can be reached with roughly the same, relatively high probability from some vertex s (where s need not necessarily be on the same side of the cut). In the special case where $H = G$ and G is rapidly mixing, the set S will be all of V, but in the general case it will be a subset of those vertices that are reached from s with probability that is not much smaller than that assigned by the stationary distribution (of $\mathrm{M}_{\ell_1}^{\ell_2}(H)$).

For states x and y in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ and an integer t, let $q_{x,y}(t)$ denote the probability that a random walk in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ that starts at x ends at y after t steps.

LEMMA 2. *Let H be a subgraph of G with at least $\frac{\epsilon}{4}n$ vertices, and let $\ell_1 = \Theta\left(\left(\frac{\log(n/\epsilon)}{\epsilon}\right)^3\right)$, $\ell_2 = \Theta\left(\frac{\ell_1}{\epsilon^2}\right)$, and $F = O\left(\frac{1}{\epsilon}\right)$. Then for every vertex s that is* useful *with respect to $\mathrm{M}_{\ell_1}^{\ell_2}(H)$, there exists a subset of vertices S in H, an integer t, $\ell_1/2 \le t \le \ell_1$, and a value $\beta = \Omega\left(\frac{\epsilon^2}{\log(n/\epsilon)}\right)$ such that the following hold:*

1. *The number of edges between S and the rest of H is at most $\frac{\epsilon}{2} \cdot d \cdot |S|$.*
2. *For every $v \in S$,*

$$\sqrt{\frac{1}{|S|} \cdot \frac{\beta}{|H|}} \le q_{s,v}(t) \le F \cdot \sqrt{\frac{1}{|S|} \cdot \frac{\beta}{|H|}}.$$

The proof of the lemma appears in [13].

3.4. Sufficient conditions for good partitions. In the next lemma we give sufficient conditions under which subsets of vertices can be partitioned without having many violating edges. For each $b \in \{0,1\}$ let $q_{s,v}^b(t)$ denote the probability in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ of a walk of length t starting from s, ending at v, and corresponding to a path whose length has parity b. What the lemma essentially requires is that for some fixed vertex s and subset of vertices S in H, there is a lower bound on the probability that each vertex in S is reached from s (in t steps), and there are not too many vertices v in the subset such that both $q_{s,v}^0(t)$ and $q_{s,v}^1(t)$ are large (with respect to this lower bound).

LEMMA 3. *Let H be a subgraph of G, s a vertex in H, S a subset of vertices in H, and ℓ_1 and ℓ_2 integers. Assume that for some $\alpha > 0$, $t < \ell_1$, the following hold in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$:*

1. *For every $v \in S$, $q_{s,v}(t) \ge \alpha$.*
2. *$\sum_{v,u \in S,\,(v,u)\in E}(q_{s,v}^0(t)q_{s,u}^0(t) + q_{s,v}^1(t)q_{s,u}^1(t)) < \frac{\epsilon}{c} \cdot d \cdot |S| \cdot \alpha^2$ for some constant c.*

Let (S_0, S_1) be a partition of S, where $S_0 = \{v : q_{s,v}^0(t) \ge q_{s,v}^1(t)\}$ and $S_1 = \{v : q_{s,v}^1(t) > q_{s,v}^0(t)\}$. Then the number of violating edges in G with respect to (S_0, S_1) is at most $\frac{\epsilon}{c} \cdot d \cdot |S|$.

Proof. Consider a vertex v and let $v \in S_b$ for $b \in \{0,1\}$. By definition of the partition (S_0, S_1), $q_{s,v}^b(t) \ge \frac{1}{2}q_{s,v}(t) \ge \frac{\alpha}{2}$.

Assume, contrary to what is claimed in the lemma, that the number of violating edges with respect to (S_0, S_1) is more than $\frac{\epsilon}{c} \cdot d \cdot |S|$. Then

$$\sum_{v,u \in S,\,(v,u)\in E} (q_{s,v}^0(t)q_{s,u}^0(t) + q_{s,v}^1(t)q_{s,u}^1(t))$$

$$\ge \sum_{v,u \in S,\,(v,u)\in E,\,u,v \in S_0} (q_{s,v}^0(t)q_{s,u}^0(t)) + \sum_{v,u \in S,\,(v,u)\in E,\,u,v \in S_1} (q_{s,v}^1(t)q_{s,u}^1(t)) \tag{3}$$

$$\ge \sum_{v,u \in S,\,(v,u)\in E,\,u,v \in S_0} \frac{\alpha^2}{4} + \sum_{v,u \in S,\,(v,u)\in E,\,u,v \in S_1} \frac{\alpha^2}{4} \tag{4}$$

$$\geq \frac{\alpha^2}{4} \cdot \frac{\epsilon}{c} \cdot d \cdot |S|. \tag{5}$$

But this contradicts the second hypothesis of the lemma. □

3.5. Sufficient conditions for detecting odd cycles. In the next lemma we describe sufficient conditions for "detecting" odd cycles when performing walks in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ starting from some vertex s. What the lemma essentially requires is that there exists a subset S of vertices such that there are both lower and upper bounds on the probability that each vertex in S is reached from s (in $t < \ell_1$ steps), and there are many vertices v in S such that both $q_{s,v}^0(t)$ and $q_{s,v}^1(t)$ are large (with respect to the lower bound). As stated later in Corollary 4, these conditions are sufficient for detecting odd cycles when performing random walks in G of length $\ell_1 \cdot \ell_2$.

LEMMA 4. *Let H be a subgraph of G, s a vertex in H, S a subset of vertices in H, and ℓ_1 and ℓ_2 integers. Assume that for some $\alpha > 0$ and $F = \Theta(\frac{1}{\epsilon})$ and $t < \ell_1$, the following hold in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$:*

1. *For every $v \in S$, $\alpha \leq q_{s,v}(t) \leq F \cdot \alpha$;*
2. *$\sum_{v,u \in S,\,(v,u)\in E}(q_{s,v}^0(t)q_{s,u}^0(t)+q_{s,v}^1(t)q_{s,u}^1(t)) \geq \frac{\epsilon}{c}\cdot d\cdot|S|\cdot\alpha^2$ for some constant c.*

Suppose we perform $O\left(\frac{F^5}{\epsilon\cdot\alpha\sqrt{d}\sqrt{|S|}}\right)$ random walks of length t starting from s in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$. Let A_0 (A_1) be the set of vertices that appear at the end of the walks whose corresponding paths have even (odd) length, and let G_0 (G_1) be the subgraph induced by A_0 (A_1). Then with probability at least 0.99 (taken over the random walks), either G_0 contains an edge or G_1 contains an edge (i.e., the algorithm detects an odd cycle).

We note that when we apply Lemma 4, we set $\alpha = \mathrm{poly}(\epsilon/(\log n))/\sqrt{|S| \cdot |H|}$ and $F = O(1/\epsilon)$ so that the number of random walks that should be performed is $O(\sqrt{n/d} \cdot \mathrm{poly}((\log n)/\epsilon))$.

Proof. Let $\gamma \stackrel{\mathrm{def}}{=} \sum_{v,u\in S,\,(v,u)\in E}(q_{s,v}^0(t)q_{s,u}^0(t)+q_{s,v}^1(t)q_{s,u}^1(t))$ so that by the second hypothesis of the lemma $\gamma \geq \frac{\epsilon}{c} \cdot d \cdot |S| \cdot \alpha^2$. Consider $m = O\left(\frac{F^5}{\epsilon\cdot\alpha\cdot\sqrt{d}\sqrt{|S|}}\right)$ random walks of length t starting from s. For $1 \leq i \neq j \leq m$, let $\eta_{i,j}$ be a 0/1 random variable that is 1 if and only if both the ith and the jth walk have path length with the same parity and if the endpoints of the paths are the vertices $u, v \in S$ such that $(u, v) \in E$.

Thus, we would like to bound the probability that $\sum_{i<j} \eta_{i,j} = 0$. The difficulty is that the $\eta_{i,j}$'s are not pairwise independent. Yet, since the sum of the covariances of the dependent $\eta_{i,j}$'s is quite small, Chebyshev's inequality is still very useful (cf. [5, sec. 4.3]). Details follow. For every $i \neq j$,

$$\mathrm{Exp}[\eta_{i,j}] \;=\; \sum_{v,u\in S,\,(v,u)\in E} (q_{s,v}^0(t)q_{s,u}^0(t) + q_{s,v}^1(t)q_{s,u}^1(t)) \;=\; \gamma.$$

By Chebyshev's inequality,

$$\Pr\left[\sum_{i<j} \eta_{i,j} = 0\right] \;\leq\; \frac{\mathrm{Var}\left[\sum_{i<j} \eta_{i,j}\right]}{\left(\mathrm{Exp}\left[\sum_{i<j} \eta_{i,j}\right]\right)^2} \;<\; \frac{\mathrm{Var}\left[\sum_{i<j} \eta_{i,j}\right]}{\left(\binom{m}{2} \cdot \gamma\right)^2}. \tag{6}$$

We now bound $\mathrm{Var}[\sum_{i<j} \eta_{i,j}]$. Since the $\eta_{i,j}$'s are not pairwise independent, some

care is needed: Let $\bar{\eta}_{i,j} \stackrel{\text{def}}{=} \eta_{i,j} - \text{Exp}[\eta_{i,j}]$.

$$\begin{aligned}
\text{Var}\left[\sum_{i<j}\eta_{i,j}\right] &= \text{Exp}\left[\left(\sum_{i<j}\bar{\eta}_{i,j}\right)^2\right] \\
&= \sum_{i<j}\sum_{k<\ell}\text{Exp}\left[\bar{\eta}_{i,j}\cdot\bar{\eta}_{k,\ell}\right] \\
&= \sum_{i<j}\text{Exp}\left[\bar{\eta}_{i,j}^2\right] + 4\sum_{i<j<k}\text{Exp}\left[\bar{\eta}_{i,j}\cdot\bar{\eta}_{j,k}\right] + 0 \\
&= \binom{m}{2}\cdot\text{Exp}[\bar{\eta}_{1,2}^2] + 4\cdot\binom{m}{3}\cdot\text{Exp}\left[\bar{\eta}_{1,2}\cdot\bar{\eta}_{2,3}\right]. \qquad (7)
\end{aligned}$$

The factor of 4 in the third equality is the number of possibilities that among the four elements i, j, k, ℓ (where $i < j$ and $k < \ell$) exactly two are equal (namely, $i = k < j < \ell$, $i < j = k < \ell$, $i < k < j = \ell$, and $k < i = \ell < j$). The 0 term is due to the fact that when i, j, k, l are all distinct,

$$\begin{aligned}
\text{Exp}\left[\bar{\eta}_{i,j}\cdot\bar{\eta}_{k,\ell}\right] &= \text{Exp}\left[\eta_{i,j}\cdot\eta_{k,\ell}\right] - \gamma^2 \\
&= \sum_{i,j,k,\ell\in S,\,(i,j),(k,\ell)\in E} q^0_{s,i}(t)q^0_{s,j}(t)q^0_{s,k}(t)q^0_{s,\ell}(t) \\
&\quad + \sum_{i,j,k,\ell\in S,\,(i,j),(k,\ell)\in E} q^0_{s,i}(t)q^0_{s,j}(t)q^1_{s,k}(t)q^1_{s,\ell}(t) \\
&\quad + \sum_{i,j,k,\ell\in S,\,(i,j),(k,\ell)\in E} q^1_{s,i}(t)q^1_{s,j}(t)q^0_{s,k}(t)q^0_{s,\ell}(t) \\
&\quad + \sum_{i,j,k,\ell\in S,\,(i,j),(k,\ell)\in E} q^1_{s,i}(t)q^1_{s,j}(t)q^1_{s,k}(t)q^1_{s,\ell}(t) - \gamma^2 \\
&= \left(\sum_{(i,j)\in E(S)} q^0_{s,i}(t)q^0_{s,j}(t) + q^1_{s,i}(t)q^1_{s,j}(t)\right)^2 - \gamma^2 = \gamma^2 - \gamma^2 = 0. \qquad (8)
\end{aligned}$$

We next bound each of the two terms in (7).

$$\text{Exp}[\bar{\eta}_{1,2}^2] \;\le\; \text{Exp}[\eta_{1,2}^2] \;=\; \text{Exp}[\eta_{1,2}] \;=\; \gamma. \qquad (9)$$

Let v_i be a random variable that represents the vertex at which the ith walk ends.

$$\begin{aligned}
\text{Exp}[\bar{\eta}_{1,2}\cdot\bar{\eta}_{2,3}] &\le \text{Exp}[\eta_{1,2}\cdot\eta_{2,3}] \\
&\le \sum_{v_1,v_2,v_3\in S,\,(v_1,v_2),(v_3,v_2)\in E} q^0_{s,v_1}(t)q^0_{s,v_2}(t)q^0_{s,v_3}(t) + q^1_{s,v_1}(t)q^1_{s,v_2}(t)q^1_{s,v_3}(t) \\
&\le (\text{number of pairs of edges in } S \text{ with a common vertex in } S) \\
&\quad \cdot 2(\max_v\{q_{s,v}(t)\})^3 \\
&\le 2\cdot\min(|S|^2 d, |S|d^2)\cdot F^3\cdot\alpha^3. \qquad (10)
\end{aligned}$$

Since by the lemma's second hypothesis $\gamma \ge \frac{\epsilon}{c}\cdot d\cdot|S|\cdot\alpha^2$, we can replace α in (10) with $\sqrt{\frac{c\cdot\gamma}{\epsilon\cdot d\cdot|S|}}$ and get

$$\text{Exp}[\bar{\eta}_{1,2}\cdot\bar{\eta}_{2,3}] \;\le\; 2\cdot\min(|S|^2 d, |S|d^2)\cdot F^3\cdot\left(\frac{c\gamma}{\epsilon d|S|}\right)^{\frac{3}{2}}. \qquad (11)$$

Combining (6)–(11), we get

$$\Pr\left[\sum_{i<j}\eta_{i,j}=0\right] = O\left(\frac{m^2\cdot\gamma + m^3\cdot\min(|S|^2 d, |S|d^2)\cdot F^3\cdot(\frac{\gamma}{\epsilon d|S|})^{\frac{3}{2}}}{m^4\cdot\gamma^2}\right)$$

$$= O\left(\frac{1}{\gamma\cdot m^2} + \frac{F^3\min\left(\sqrt{\frac{|S|}{d}}, \sqrt{\frac{d}{|S|}}\right)}{m\cdot\epsilon^{\frac{3}{2}}\cdot\sqrt{\gamma}}\right)$$

$$= O\left(\frac{\epsilon}{F^{10}} + \frac{1}{F^2\cdot\epsilon}\right) = O(\epsilon). \tag{12}$$

As observed above, by the lemma's hypothesis concerning γ, it holds that $\alpha = O(\sqrt{\gamma/(\epsilon d|S|)})$. Since $m = \Omega\left(\frac{F^5}{\epsilon\cdot\alpha\cdot\sqrt{d}\sqrt{|S|}}\right)$, we have that $m = \Omega\left(F^5\sqrt{\frac{1}{\epsilon\cdot\gamma}}\right)$, and the lemma follows. □

Based on the construction of $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ we can map walks of length $\ell_1\cdot\ell_2$ in G to walks of length ℓ_1 in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ and obtain the following corollary to Lemma 4.

COROLLARY 4. *Let H be a subgraph of G, and let S, s, ℓ_1, ℓ_2, t, α, and F be as in Lemma 4. Suppose we perform $\Theta\left(\frac{F^5}{\epsilon\cdot\alpha\cdot\sqrt{d}\sqrt{|S|}}\right)$ random walks of length $\ell_1\cdot\ell_2$ starting from s in G. Let A_0 (A_1) be the set of vertices that appear at the end of the walks whose corresponding paths have even (odd) length, and let G_0 (G_1) be the subgraph induced by A_0 (A_1). Then with probability at least 0.99, either G_0 contains an edge or G_1 contains an edge (i.e., the algorithm detects an odd cycle).*

The proof of the corollary is similar to that of an analogous corollary that appears in [13].

3.6. Proof of Theorem 1. Recall that we need to show that if the test accepts G with probability greater than $\frac{1}{3}$, then G is ϵ-close to being bipartite.

We say that a vertex s in G is *good* (for defining a partition) if the following holds. Suppose we take K random walks of length L in G starting from s. Then the probability that we reach two vertices u and v such that $(u,v)\in E$ and both u and v appear at the ends of walks whose corresponding paths have lengths with the same parity is at most 0.1. If a vertex is not good, then it is *bad*. Here K and L are set in the algorithm.

Since the test rejects G with probability less than $\frac{2}{3}$, and $T=\Theta(1/\epsilon)$, we know that, for an appropriate constant in the $\Theta(\cdot)$ notation above, the fraction of bad vertices in G is at most $\frac{\epsilon}{16}$. We now show that in such a case we can find a partition of the graph vertices that has at most ϵdn violating edges. We shall do so in steps, where in each step we partition a new set of vertices, denoted S, until we are left with at most $\frac{\epsilon}{4}n$ vertices. For each partitioned set S we show that (1) there are few (at most $\frac{\epsilon}{4}d|S|$) violating edges with respect to the partition of S; and (2) there are few (at most $\frac{\epsilon}{2}d|S|$) edges between S and the yet "unpartitioned" vertices R so that no matter how the vertices in R are partitioned, the number of violating edges between S and R is small.

At each step, let D be the set of vertices we have already partitioned, and let H be the subgraph induced by $V\setminus D$. Initially, $D=\emptyset$, and $H=G$. Let ℓ_1 and ℓ_2 be as required by Lemma 2, and let the length L of the walks we perform on G be $\ell_1\cdot\ell_2$. Since $\ell_1 = O\left(\left(\frac{\log(n/\epsilon)}{\epsilon}\right)^3\right)$, and $\ell_2 = O\left(\frac{\ell_1}{\epsilon^2}\right)$, we get that $L = O\left(\frac{\log^6(n/\epsilon)}{\epsilon^8}\right)$. Let $\mathrm{M} \stackrel{\text{def}}{=} \mathrm{M}_{\ell_1}^{\ell_2}(H)$. While $|H|\geq\frac{\epsilon}{4}n$, we do the following. We select any vertex s in H

that is both *good* and *useful* with respect to M (see Definition 3.1). By Lemma 1, at least half of the vertices in H are *useful*. Since $|H| \geq \frac{\epsilon}{4}n$ and the total number of *bad* vertices is $\frac{\epsilon}{16}n < \frac{\epsilon}{8}n$, there exist at least $\frac{\epsilon}{16}n$ vertices which are *good* and *useful*.

We next apply Lemma 2 to determine a set S and an integer t, $\ell_1/2 \leq t \leq \ell_1$, with the properties stated in the lemma. In particular, the number of edges between S and the rest of H is at most $\frac{\epsilon}{2}d|S|$, and for every $v \in S$, $\sqrt{\frac{\beta}{|S|\cdot|H|}} \leq q_{s,v}(t) \leq F \cdot \sqrt{\frac{\beta}{|S|\cdot|H|}}$, where $F = O\left(\frac{1}{\epsilon}\right)$, and $\beta = \Omega\left(\frac{\epsilon^2}{\log(n/\epsilon)}\right)$. We claim that it must be the case that $\sum_{v,u\in V,\,(v,u)\in E}(p_v^0(t)p_u^0(t) + p_v^1(t)p_u^1(t)) \leq \frac{\epsilon\cdot\beta\cdot d}{|H|}$. This claim (which we establish momentarily) implies that we can apply Lemma 3 (with $\alpha = \sqrt{\frac{\beta}{|S|\cdot|H|}}$) to show that S can be partitioned so that there are at most $\frac{\epsilon}{4}d|S|$ violating edges with respect to this partition. The claim holds since otherwise we could apply Corollary 4 and reach a contradiction. Specifically, by letting the number of walks performed from each starting vertex be

$$O\left(\frac{F^5}{\epsilon\cdot\alpha\cdot\sqrt{d}\cdot\sqrt{|S|}}\right) = O\left(\frac{\sqrt{|H|}}{\epsilon^6\cdot\sqrt{d}\cdot\sqrt{\beta}}\right) = O\left(\frac{\log^{1/2}(n/\epsilon)\cdot\sqrt{n/d}}{\epsilon^8}\right) = K$$

(where F, α, and β are as set above), we would obtain a contradiction to our assumption that s is *good*.

Thus, as long as $|H| \geq \frac{\epsilon}{4}n$, each set S contributed at most $\frac{\epsilon}{4}\cdot|S|\cdot d + \frac{\epsilon}{2}\cdot|S|\cdot d$ violating edges to the partition. Since these sets are disjoint, all these violating edges sum up to $\frac{3\epsilon}{4}\cdot d\cdot n$. The final H contributes at most $\frac{\epsilon}{4}\cdot n\cdot d$, and so G is ϵ-close to being bipartite.

Verifying that indeed $T = O(1/\epsilon)$, $K = \Theta(\sqrt{n/d}\cdot\mathrm{poly}(\log n/\epsilon))$, and $L = \mathrm{poly}((\log n)/\epsilon))$ and that the algorithm can be implemented using $O(K\cdot L + K^2) = O(n/d\cdot\mathrm{poly}(\log n/\epsilon))$ queries, the theorem follows. (Recall that if $d < \sqrt{n}$, then we obtain the bound of $O(\sqrt{n}\cdot\mathrm{poly}(\log n/\epsilon))$.)

4. The algorithm for the general case. In this section we build on the testing algorithm presented in the previous section and describe a one-sided error testing algorithm for bipartiteness that works with respect to the actual number of edges $m = m(G)$. Hence this algorithm is suitable for general graphs (for which $d_{\max}$ may vary significantly from d_{avg}). The query complexity and running time of the algorithm are of the same order of magnitude as for Test-Bipartite-Reg, that is, $O(\min(\sqrt{n}, n^2/m)\cdot\mathrm{poly}(\log n/\epsilon))$. We note that once the graph becomes very dense, that is, $m = \Omega(n^2/\log^c n)$ (where c is approximately 4), it is preferable to use the adjacency-matrix model algorithm [12, 3] with distance parameter $\epsilon/(n^2/m)$.

A high level description of the algorithm. The basic idea is to reduce the problem of testing with respect to the actual number of edges m to the problem of testing with respect to the upper bound $m_{\max} = d_{\max}\cdot n$. Specifically, for any graph G we show how to define a graph G' over $\Theta(n)$ vertices that has the following useful properties. First, the maximum degree in G' is roughly the same as the average degree, and furthermore, this degree is roughly the same as the average degree in G. In particular, this implies that the two graphs have roughly the same number of edges. Second, G' approximately preserves the distance of G to bipartiteness. More precisely, if G is bipartite, then so is G', but if G is far from being bipartite with respect to $m(G)$, then G' is far from being bipartite with respect to $m_{\max} = d_{\max}(G')n'$. Thus G' can be viewed as a kind of "regularized-degree version" of G.

If we had direct access to G', then by the above we would be done: by running the algorithm Test-Bipartite-Reg on G' we could decide whether G is bipartite or far from being bipartite. However, we have access only to G. Nonetheless, given query access to G we can efficiently "emulate" queries in G'. This would almost suffice for running Test-Bipartite-Reg on G'. One more issue is the uniform selection of starting vertices in G', required by Test-Bipartite-Reg. As we shall see, selecting a vertex uniformly from G' is (roughly) equivalent to uniformly selecting an edge in G.

While we do not know how to efficiently select a vertex in G' uniformly, we describe a different selection procedure that suffices for our purposes. Specifically, the selection procedure is such that for all but a small fraction of the n' vertices in G', the probability of selecting a vertex v is $\Omega(1/n')$. With slight abuse of terminology we shall refer to this procedure as Sample-Vertices-Almost-Uniformly-in-G'.[3] By Corollary 3, this suffices for our purposes.

Multiple edges and the relation between m and n. The analysis of the algorithm Test-Bipartite-Reg did not require any assumptions on the actual number of edges m in the graph, and it did not preclude the existence of multiple edges. Here we consider graphs that do not contain any multiple edges, and we assume that the number of edges m in G is $\Omega(n)$. To justify the assumption on the number of edges, consider a graph that consists of a clique over $k = o(\sqrt{n})$ vertices, where all remaining vertices are isolated. This graph has $m = \Theta(k^2)$ edges and is clearly far from being bipartite. However, in order to distinguish it from a graph that consists of a complete bipartite graph over $2k$ vertices (where all remaining vertices are isolated and clearly bipartite), we need $\Omega(n/k) = \omega(\sqrt{N})$ queries. (Taking this to an extreme, if $k = \Theta(1)$, then we will need $\Omega(n)$ queries.) We note that we could replace this assumption by introducing to the complexity of the algorithm a dependence on n/m. This would, however, make the analysis more cumbersome, without much benefit.

Another alternative assumption would be that the algorithm has the ability to "ignore" isolated vertices (that is, vertices that have no incident edges and are hence immaterial to the question of bipartiteness) and sample uniformly from the *nonisolated* vertices. This would effectively imply that the algorithm is executed on a subgraph induced by the $n' \leq n$ nonisolated vertices, where within this subgraph, the number of edges $m' = m$ is at least $n'/2$.

For simplicity we assume from this point on that $m \geq n$.

We also note that we can actually deal with the case where there are multiple edges, but they do not constitute more than a constant fraction of the total number of edges.[4] However, in order to deal with this case efficiently, we need to assume that there is a concise way to represent the sets of labels of multiple edges that are incident to each vertex. (In particular, this holds if the labels of multiple edges incident to each vertex are consecutive.) For simplicity we assume there are no multiple edges.

The main theorem of this subsection follows.

THEOREM 5. *For every graph G having n vertices and $m \geq n$ edges, we can define a graph G' having n' vertices and m' edges for which the following hold:*

[3]The reason we say that we abuse terminology is that the distribution on vertices in G' induced by this procedure may be very far from uniform according to any standard distance measure (e.g., statistical difference). However, it approximates the uniform distribution in the sense of assigning relatively large weight to every sufficiently large subset.

[4]If the number of multiple edges is more than a constant fraction, then it is possible to obtain a lower bound on the number of queries that depends on the ratio between the number of multiple edges and the total number of edges. Specifically, consider a graph that contains a small clique with many multiple edges, which is far from bipartite but cannot be distinguished from a bipartite graph that contains a small complete bipartite graph with many multiple edges.

1. $n \leq n' \leq 4n$, $m \leq m' \leq 8m$, *and* $d_{\max}(G') \leq 2d_{\text{avg}}(G)$.
2. *If* G *is bipartite, then* G' *is bipartite, and if* G *is* ϵ*-far from being bipartite with respect to* m*, then* G' *is* ϵ'*-far being from bipartite with respect to* $m_{\max}(G') = d_{\max}(G')n'$ *for* $\epsilon' = \Theta(\epsilon)$.
3. *Given a starting vertex* s *in* G' *it is possible to emulate random walks in* G' *starting from* s *by performing queries to* G. *The amortized cost of each random-walk step is* $O(\log^2 n)$ *(degree and neighbor) queries in* G. *By emulating these random walks it is possible to execute a slight variant of* Odd-Cycle(s) *in* G' *which we denote* Odd-Cycle$'(s)$. *This variant is such that* $\Pr[\text{Odd-Cycle}'(s) = \textsf{found}] \geq \Pr[\text{Odd-Cycle}(s) = \textsf{found}]$, *where if* Odd-Cycle$'(s)$ *returns* found, *then we can obtain an odd cycle of length* $\text{poly}(\log n/\epsilon)$ in the original graph G.
4. *There exists a procedure* Sample-Vertices-Almost-Uniformly-in-G' *that for any given parameter* $0 < \delta \leq 1$ *performs* $\tilde{O}(\min(\sqrt{n/\delta}, n^2/m))$ *queries in* G *and returns a vertex in* G' *such that the following holds: For all but at most* $\delta n'$ *of the vertices* x *in* G', *the probability that* x *is selected by the procedure is* $\Omega(1/n')$.

We note that for every graph G there is actually a *family* of graphs G' with the above properties (all defined over the same set of vertices). When we run algorithm Test-Bipartite-Gen, we construct one such (arbitrary) graph G' in the family as we go along. One difficulty that arises is that when the algorithm asks a neighbor query of the form "Who is the ith neighbor of v?", it gets a vertex name u as an answer. However, the algorithm lacks the information that v is (say) the jth neighbor of u. This lack of knowledge makes the emulation of random walks and Odd-Cycle(s) in G' more complicated. Due to that, item 3 of Theorem 5 is somewhat more involved.

As a corollary to Theorem 5 and Corollary 3 we obtain the following.

COROLLARY 6. *Algorithm Test-Bipartite-Gen (see Figure* 2*) accepts every graph* G *that is bipartite and rejects with probability at least* $2/3$ *every graph* G *that is* ϵ*-far from being bipartite (with respect to* $m(G)$*). Furthermore, whenever the algorithm rejects a graph, it outputs a* certificate *to the nonbipartiteness of the graph* G *in the form of an odd cycle of length* $\text{poly}(\log n/\epsilon)$. *The query complexity and running time of the algorithm are* $O\left(\min(\sqrt{n}, n^2/m) \cdot \text{poly}(\log n/\epsilon)\right)$.

Test-Bipartite-Gen($n, d_{\text{avg}}, \epsilon$)

- Repeat $T = \Theta(\frac{1}{\epsilon})$ times:
  1. Set $\epsilon' = \epsilon/144$.
  2. Select a vertex s in G' by calling the procedure Sample-Vertices-Almost-Uniformly-in-G' with $\delta = \epsilon'/c$ (where c is a sufficiently large constant).
  3. Apply Odd-Cycle$'(s)$.
  4. If Odd-Cycle$'(s)$ returns found, then output reject.
- In case no call to Odd-Cycle$'$ returned found, then output accept.

FIG. 2. *Algorithm Test-Bipartite-Gen for testing bipartiteness with respect to the actual number of edges* $m = m(G)$ *in the graph* G.

Note that d_{avg}, the average degree of the graph, is given as a parameter to the algorithm. Since the algorithm does not actually need the exact value of d_{avg}, it can instead estimate it. Specifically, Feige [11] shows how to obtain an estimate that is within a factor of roughly 2 of the true value by performing at most $O(\sqrt{n/d_0})$ degree queries where d_0 is an a priori lower bound on d_{avg}. Inspired by our procedure

Sample-Edges-Almost-Uniformly-in-G, Goldreich and Ron [15] suggest an alternative procedure that improves on the quality of the estimate. More importantly in our context, they observe that both in the case of their procedure and in the case of Feige's procedure, it is possible to eliminate the need of the lower bound d_0. That is, given's Feige's algorithm, it is possible to obtain a constant factor estimate by performing $O(\sqrt{n/d_{\text{avg}}}) \leq O(\min(\sqrt{n}, n^2/m))$ queries (but without any knowledge about d_{avg}).

We now turn to proving Theorem 5. We actually provide two proofs: one is based on a deterministic construction of G' and one on a probabilistic construction. We start by presenting the deterministic construction.

4.1. Defining G' and proving the first two items in Theorem 5. In all that follows, let $d = d_{\text{avg}}(G)$, and let $d' = d_{\max}(G')$. We shall assume that d is a sufficiently large constant. If $d_{\text{avg}}(G)$ is not sufficiently large, then we still set d in the construction below to be sufficiently large and run the algorithm with ϵ set to $\epsilon/(d/d_{\text{avg}}(G))$.

The idea. Recall that part of our goal is to have G' be a "regularized" version of G in the sense that in G' all vertices have degree at most $2d$ (while in G there may be vertices with degree much higher than the average degree d). To this end, every vertex of G with degree higher than d is represented in G' by a subset of vertices. Each such subset is partitioned into two equal parts: an *external* subset (consisting of *external* vertices) and an *internal* subset (consisting of *internal* vertices). The edges between external vertices in G' are determined by the edges of G. Namely, if (u, v) is an edge in G, then in G' there is an edge between one of the vertices in the external subset of u to one of the vertices in the external subset of v. In addition, for every vertex v (with degree greater than d) there is a bipartite subgraph between its internal and external vertices. All vertices in the subgraph have degree d, and the subgraph has good expansion properties.

The role of these subgraphs between external and internal vertices is to ensure that if G is far from being bipartite, then so is G'. To gain some intuition observe that for every partition (V_1, V_2) of G, there exists a corresponding partition (V_1', V_2') of G' that has the same number of violating edges. Specifically, for every vertex $v \in V_1$ ($v \in V_2$) we put in V_1' (V_2') all external vertices that correspond to v, and we put in V_2' (V_1') all internal vertices that correspond to v. By this construction, there is a one-to-one mapping between the edges in G that are violating with respect to (V_1, V_2) and the edges in G' that are violating with respect to (V_1', V_2') (where all edges in the subgraphs between external and internal vertices are nonviolating). Thus, if G is far from being bipartite, so that all partitions (V_1, V_2) of G have many violating edges, then this is also true of all partitions (V_1', V_2') as defined above. However, there are other partitions of G' that may split the external vertices that correspond to the same vertex in G into different parts and possibly have fewer violating edges. What we show is that such splits must introduce violating edges in the subgraphs between external and internal edges, where this is due to the expansion properties of the subgraphs. Roughly speaking, if a partition "avoids violations" between external vertices by splitting sets of external vertices, then it "pays" by introducing violations between external and internal vertices.

4.1.1. The construction of G'. For each vertex v in G such that $\deg(v) \leq d$, we have a single vertex in G'. For each vertex v in G such that $\deg(v) > d$, the graph G' contains a subgraph, denoted $H(v)$. It is a bipartite graph over two subsets of vertices, one denoted $X(v)$, the *external* part, and one denoted $I(v)$, the *internal*

part. Both parts consist of $\lceil \deg(v)/d \rceil$ vertices. Every vertex in $X(v)$ is assigned up to d specific neighbors of v according to some fixed *but arbitrary* partition of the neighbors of v. As we shall see below, this assignment determines the edges between pairs of external vertices in G' that correspond to different vertices in G. We refer to the vertices in the two subsets by $\{X_i(v)\}_{i=1}^{\lceil \deg(v)/d \rceil}$ and $\{I_i(v)\}_{i=1}^{\lceil \deg(v)/d \rceil}$, respectively.

The edges in $H(v)$ are determined as follows. In case $\deg(v)/d < d$ then we have $\frac{\lfloor d^2/\deg(v) \rfloor}{2}$-multiple edges between every internal vertex and every external vertex in $H(v)$. It follows that the degree of every vertex within $H(v)$ is

$$\frac{\lfloor d^2/\deg(v) \rfloor}{2} \cdot \lceil \deg(v)/d \rceil \;\leq\; \frac{d^2}{2 \cdot \deg(v)} \cdot \frac{\deg(v)+d}{d} \;\leq\; \frac{d^2}{2 \cdot \deg(v)} \cdot \frac{2\deg(v)}{d} \;=\; d.$$

In case $\deg(v)/d \geq d$, denote $s = \lceil \deg(v)/d \rceil$, and let $H(v)$ be a bipartite expander where each of its sides has s vertices ($s \geq d$). Each vertex in $H(v)$ has degree d. All eigenvalues of the adjacency matrix of H but the largest one and the smallest one (which are equal to d and $-d$, respectively) are at most $d/4$ in their absolute values. Explicit constructions of such expanders can be found, e.g., in [18, 17]. Furthermore, these constructions allow the determination of the ith neighbor of any given vertex in constant time.

For the sake of the presentation, when $\deg(v) \leq d$ so that v is represented by a single vertex, we let $H(v)$ be the subgraph that consists of this single vertex. This vertex is considered an external vertex, denoted $X_1(v)$, and it is assigned all neighbors of v.

We have described how vertices of G are transformed into vertices of G' (some of which are connected by edges). It remains to describe the relevant transformation to the edges of G. Consider an edge $(u,v) \in E(G)$, where v is the ith neighbor of u and u is the jth neighbor of v. Let $X_k(u)$ and $X_\ell(v)$ be the external vertices that are assigned the ith neighbor of u and the jth neighbor of v, respectively. Then, there is an edge $(X_k(u), X_\ell(v))$ in G'. It directly follows that every vertex in G' has degree at most $2d$ and that $n' = |V(G')| \leq \sum_{v \in G} 2\lceil \deg(v)/d \rceil \leq 4n$, and $m' = m(G') \leq 4dn = 8m$.

For an illustration of the construction of G', see Figure 3.

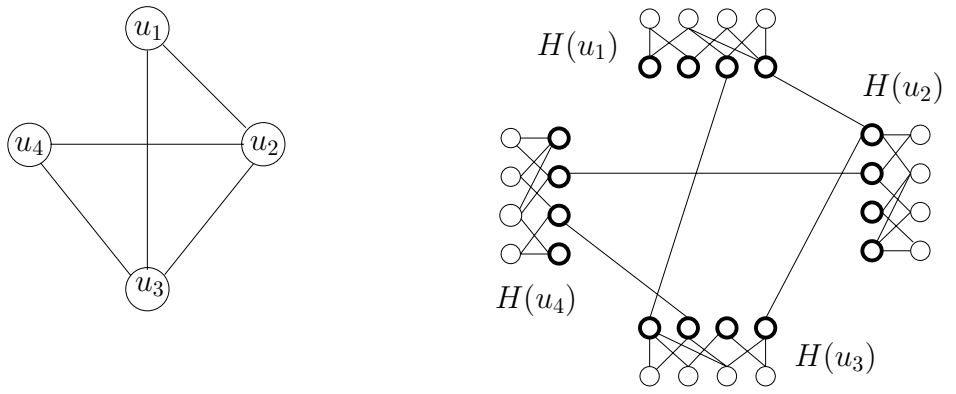


FIG. 3. *An illustration for the construction of G'. On the left are four vertices in G and their induced subgraph. On the right are the four corresponding subgraphs in G' and the edges between the external vertices in these subgraphs. (The external vertices are marked in bold, and there are additional edges that do not appear in the figure between the external vertices in the figure and external vertices of other subgraphs.)*

We have thus established the first item in Theorem 5, and we turn to the second item. From the construction of G' it is obvious that if G is bipartite, then so is G'. It remains to prove the following lemma.

LEMMA 5. *If G is ϵ-far from being bipartite (with respect to $m = (dn)/2$), then G' is ϵ'-far from being bipartite with respect to $d'n'$ for $\epsilon' = \frac{\epsilon}{144}$.*

In order to prove Lemma 5, we first prove the following proposition concerning bipartite expander graphs. For any two (disjoint) subsets of vertices, A and B, we let $e(A, B)$ denote the number of edges with one endpoint in A and another in B.

PROPOSITION 7. *Let $G = (A \cup B, E)$ be a d-regular bipartite graph with sides A and B of size s. Assume that all eigenvalues of the adjacency matrix of G, but the largest one and the smallest one, are at most λ in their absolute values. Assume further that $\lambda \leq d/4$. Then for every two partitions $A = A_1 \cup A_2$, $B = B_1 \cup B_2$, satisfying $|A_1| \geq s/2$,*

$$e(A_1, B_1) + e(A_2, B_2) \geq \frac{d|A_2|}{8}.$$

Proof. It is well known that the larger the "spectral gap" (i.e., the difference between d and λ) is, the closer the edge distribution in G approaches that of a truly random bipartite graph with sides of size s and edge probability d/s. Specifically, for every $A_0 \subseteq A$, $B_0 \subseteq B$ of sizes $|A_0| = a_0$, $|B_0| = b_0$,

$$\left| e(A_0, B_0) - \frac{da_0b_0}{s} \right| \leq \lambda\sqrt{a_0b_0} \tag{13}$$

(see, e.g., Chapter 9 of [5]).

Let $|A_1| = a_1$, $|A_2| = a_2 = s - a_1$, $|B_1| = b_1$, $|B_2| = b_2 = s - b_1$. It is given that $a_1 \geq s/2$. We may obviously assume that $b_1 \geq a_2/2$, as otherwise at least half of the edges incident to A_2 have their other endpoint outside B_1, implying $e(A_2, B_2) \geq da_2/2$.

Applying the bound in (13) twice, we get

$$e(A_1, B_1) = d|B_1| - e(A_2, B_1) \geq db_1 - \frac{da_2b_1}{s} - \lambda\sqrt{a_2b_1} = \frac{da_1b_1}{s} - \lambda\sqrt{a_2b_1}, \tag{14}$$

$$e(A_2, B_2) = d|A_2| - e(A_2, B_1) \geq da_2 - \frac{da_2b_1}{s} - \lambda\sqrt{a_2b_1} = \frac{da_2b_2}{s} - \lambda\sqrt{a_2b_1}. \tag{15}$$

Consider first the case $b_2 \leq s/2$. In this case it follows from (14) that

$$\begin{aligned} e(A_1, B_1) &\geq \frac{da_1b_1}{s} - \lambda\sqrt{a_2b_1} \geq \frac{d(s/2)(s/2)}{s} - \lambda\sqrt{(s/2)s} \\ &= \frac{ds}{4} - \frac{\lambda s}{\sqrt{2}} \geq \frac{ds}{14} \geq \frac{da_2}{7}. \end{aligned}$$

We thus assume that $b_2 > s/2$. If $da_2b_2/s \geq 2\lambda\sqrt{a_2b_1}$, we obtain from (15) that

$$e(A_2, B_2) \geq \frac{da_2b_2}{2s} \geq \frac{da_2(s/2)}{2s} = \frac{da_2}{4}.$$

Hence we may assume that $da_2b_2/s \leq 2\lambda\sqrt{a_2b_1}$. If $da_1b_1/s \geq 2\lambda\sqrt{a_2b_1}$, then it follows from (14) that

$$e(A_1, B_1) \geq \frac{da_1b_1}{2s} \geq \frac{d(s/2)(a_2/2)}{2s} = \frac{da_2}{8},$$

as required. Hence we may assume that $da_1b_1/s \leq 2\lambda\sqrt{a_2b_1}$. It remains to check that the latter assumption together with $da_2b_2/s \leq 2\lambda\sqrt{a_2b_1}$ brings us to a contradiction. Indeed, multiplying these inequalities, we get $d^2a_1a_2b_1b_2/s^2 \leq 4\lambda^2a_2b_1$ or

$d^2 a_1 b_2 \leq 4\lambda^2 s^2$. Recalling that $a_1 \geq s/2$, $b_2 > s/2$, it follows that $d < 4\lambda$, which is a contradiction to our assumption on λ. □

Proof of Lemma 5. We shall show that the number of edges that should be removed from G' so as to make it bipartite is at most a constant factor smaller than the number of edges that should be removed from G so as to make G bipartite. Since the total numbers of edges in G and in G' are of the same order, this suffices to prove the lemma. To this end we prove the contrapositive statement. Specifically, suppose G' is ϵ'-close to being bipartite with respect to $m_{\max} = d'n'$. Namely, there exists a partition $P' = (V_0', V_1')$ of the vertices in G' with respect to which there are at most $\epsilon' \cdot d'n'$ violating edges in G'. We shall show how to construct, based on P', a partition $P = (V_0, V_1)$ of the vertices in G with respect to which there are at most $\epsilon dn/2 = \epsilon m$ violating edges in G, thus proving the lemma.

Consider a particular vertex v in G and the subset of external vertices $X(v)$ in G' that correspond to v. Let $X^0(v) = X(v) \cap V_0'$, and let $X^1(v) = X(v) \cap V_1'$. We refer to the larger subset as the *majority subset* of v and to the smaller subset as the *minority subset* of v. We define $P = (V_0, V_1)$ by assigning each vertex v in G according to its majority subset. Namely, if $|X^0(v)| \geq |X^1(v)|$, then v is assigned to V_0; otherwise, it is assigned to V_1. In what follows it will be convenient to refer to 0 and 1 as *colors*.

Also, when we refer to edges in G' as violating edges, we mean with respect to P', and when we refer to edges in G as violating edges, we mean with respect to P. Note that the partition P is defined only according to the coloring of the external vertices in G', ignoring the coloring of the internal vertices. Also recall that there is a one-to-one mapping between edges in G and edges in G' whose endpoints are both external vertices.

Since each vertex v in G is assigned the color of its majority subset, the violating edges in G' between pairs of vertices that both belong to majority subsets, or between pairs of vertices that both belong to minority subsets, become violating edges in G. Similarly, nonviolating edges in G' between vertices in majority subsets, or between vertices in minority subsets, become nonviolating edges in G. It remains to deal with edges between minority and majority subsets in G'. These edges can be nonviolating in G' but may become violating in G.

We next show that the total number of vertices in G' that belong to minority subsets can be bounded as a function of the number of violating edges in G'. To this end we show that if there were many minority vertices, then there would be many violating edges in G' *between internal and external vertices*.

For each vertex v in G, consider the majority and minority subsets of (the external vertices of) v. Let the majority subset of $X(v)$ be $X^\alpha(v)$, and let the minority subset be $X^\beta(v)$ (where $\alpha, \beta \in \{0,1\}$).

Claim 1. *For every vertex v in G, the number of violating edges in G' between vertices in $X(v)$ and vertices in $I(v)$ is at least $\left|X^\beta(v)\right| \cdot (d/8)$.*

Proof. Similarly to our notation for external vertices, for the internal vertices of v let $I^0(v) \stackrel{\text{def}}{=} I(v) \cap V_0'$ and $I^1(v) \stackrel{\text{def}}{=} I(v) \cap V_1'$. Consider first the case $\frac{\deg(v)}{d} < d$. By construction of G', $|X(v)| = |I(v)| = \lceil \deg(v)/d \rceil$, and there are $\frac{\lfloor d^2/\deg(v) \rfloor}{2}$ multiple edges between every pair of vertices (x, y) such that $x \in X(v)$ and $y \in I(v)$. Hence the number of edges between $X(v)$ and $I(v)$ that are violating (with respect to P') is

$$\begin{aligned}
&\left(|X^\alpha(v)||I^\alpha(v)| + |X^\beta(v)||I^\beta(v)|\right) \cdot \frac{\lfloor d^2/\deg(v) \rfloor}{2} \\
&\geq \left(|X^\beta(v)||I^\alpha(v)| + |X^\beta(v)||I^\beta(v)|\right) \cdot \frac{d^2}{4\deg(v)} = |X^\beta(v)| \cdot \frac{d}{4}.
\end{aligned}$$

Next consider the more interesting case where $\frac{\deg(v)}{d} \geq d$. In this case Claim 1 directly follows from Proposition 7. □

Thus we can conclude that if there are w external vertices that belong to minority subsets, then they contribute at least $w \cdot d/8$ violating edges in G'. Since the number of violating edges in G' is at most $\epsilon' n' d' \leq 8\epsilon' nd$, we have that $w \leq 64\epsilon' n$. As noted previously, the total number of violating edges in G is upper bounded by the number of violating edges in G' plus the number of edges between minority and majority (external) subsets. By the above discussion and the fact that every external vertex has at most d neighbors that are external vertices, the number of violating edges in G is at most $8\epsilon' nd + 64\epsilon' nd = 72\epsilon' nd$. Since $\epsilon' = \epsilon/144$ and $m = (nd)/2$, the lemma follows. □

We have completed proving the first two items of Theorem 5, and we now turn to the third item.

4.2. Establishing item 3 in Theorem 5. Here we stress that if the neighbor and the vertex-pair queries would have returned more information, then the proof of the current item would be significantly simpler. In particular, suppose that a neighbor query (u, i) is answered with a pair (v, j) (instead of only v), which means that v is the ith neighbor of u and u is the jth neighbor of v. Suppose also that when performing a vertex-pair query (u, v), the algorithm is not only told whether (u, v) is an edge or not, but rather, in the former case, it is also provided with pair (i, j), which means that v is the ith neighbor of u and u is the jth neighbor of v. With this additional information, the structure of G' is implicitly given, and thus the emulation of random walks in the execution of the procedure Odd-Cycle on G' is straightforward. Here we need to work harder to overcome the lack of information.

Random-walk steps. We first briefly discuss the emulation of random walks. If the walk stays at the current vertex, then clearly there is no need for any emulation. Hence, we need only to consider the case in which we have to select a random neighbor. Recall that vertices in G' are either of the form $X_i(v)$ (the ith external vertex corresponding to vertex v in G) or of the form $I_i(v)$ (the ith internal vertex), where $1 \leq i \leq \lceil \deg(v)/d \rceil$. Recall that we can obtain $\deg(v)$ for any v by a single degree query and, in particular, use this to find the degree of vertices in G'. For simplicity of presentation, we assume from this point on that for every vertex v in G, the degree of every vertex $I_i(v)$ in G' is d (instead of being at most d), and the degree of every $X_i(v)$ is $2d$ (instead of being at most $2d$).

Performing a random-walk step in G' from an internal vertex $I_i(v)$ can be easily done by using the explicit structure of the graph $H(v)$ (which is either a complete bipartite graph with multiple edges or an explicitly constructible expander). In order to perform a random-walk step from an external vertex, $X_i(v)$, we first determine whether to take one of the d edges within the graph $H(v)$ or whether to take one of the d edges going from $X_i(v)$ to another external vertex. In the first case we then select an internal neighbor given the explicit structure of $H(v)$. It remains to deal with selecting an external neighbor. Note that in the special, but easy, case in which $\deg(v) \leq d$ and so $H(v)$ is a single vertex, there is only the latter option.

As noted just following the statement of Theorem 5, we actually construct G' as we go along. The important thing to note is that the definition of G' allows us to assign the vertices in $X(v)$ edges of v in an *arbitrary* manner (as long as each $X_i(v)$ is assigned (at most d) different edges). Hence, all we need to take care of is to be consistent with previous choices and to ensure the correct distribution in the choice of the random-walk step. To this end we may think of each external vertex as having

d "ports," labeled $1, \ldots, d$, which are initially unassigned. As the algorithm proceeds, it puts a "link" between, say, the tth port of $X_i(v)$ and the ℓth port of $X_j(u)$ (where $(v,u) \in E(G)$). When performing a random-walk step from $X_i(v)$ (to a vertex outside of $H(v)$), we uniformly select a port. Let us denote the index of the port selected by t. If port t of $X_i(v)$ is already linked to another port, then we simply take this link to the port (and vertex) at the other end. Otherwise, we first set the link and then take it. In order to set the link properly, we need to uniformly select a neighbor u of v among the neighbors of v *that were not yet assigned* (to any port of one of the external vertices of v). After doing so and selecting a neighbor u, we need to select a yet unassigned port of one of the external vertices of u. The above can be done, with the aid of sampling, at an amortized cost of $O(\log n)$ queries in G. Details follow.

For each external vertex $X_i(v)$, the algorithm keeps a vector of length d, $\Gamma_i(v)$. The kth entry of $\Gamma_i(v)$ contains the kth neighbor of $X_i(v)$, if it was determined, and is empty otherwise. Denote by $f_i(v)$ the number of free entries in the vector $\Gamma_i(v)$. Also denote by $A(v)$ the set of neighbors of v that were already assigned to external vertices of v, and by $NA(v)$ the vertices that were not assigned. When setting a link to a yet-unassigned port (filling in a new entry in $\Gamma_i(v)$), we distinguish between two cases.

Case 1. If less than half of the neighbors of v in G are in $A(v)$, then the algorithm repeats at most $(\log^2 n)$ times the following procedure: It chooses uniformly at random a neighbor of v in G. If that neighbor belongs to $NA(v)$, then a desired neighbor is found. By repeating the above procedure $O(\log^2 n)$ times, the probability that the algorithm did not find a desired neighbor is at most $o(1/n)$. In this case we say that the algorithm fails. Since the total number of queries that the algorithm performs is at most $o(n)$, the total failure probability of the algorithm is $o(1)$. Suppose that a desired neighbor of v with the name u is found. In that case the algorithm should move to one of the external vertices of u. The selected vertex $X_k(u)$ is chosen with probability $\frac{f_k(u)}{\sum_{1 \le j \le \lceil \deg(u)/d \rceil} f_j(u)}$. According to the chosen $X_j(u)$, the algorithm sets $\Gamma_i(v)[t] \leftarrow X_j(u)$. In addition, the algorithm chooses uniformly at random one of the $f_j(u)$ free entries in the vector $\Gamma_j(u)$. Assume that the chosen index is t', $1 \le t' \le d$; then the algorithm sets $\Gamma_j(u)[t'] \leftarrow X_i(v)$.

Case 2. If more than half of the neighbors of v in G are in $A(v)$, then the algorithm reads all the neighbors of v in G that belong to $NA(v)$ and attaches them arbitrarily to the unoccupied entries in $\Gamma_j(v)$, $1 \le j \le \lceil \deg(v)/d \rceil$. By doing this, the algorithm (at most) doubles the number of neighbor queries performed on vertex v of G. Now, suppose that in $\Gamma_i(v)[t]$ there is a name of a vertex of G, say, u. In that case, the algorithm should move to one of the external vertices $X_k(u)$, $1 \le k \le \lceil \deg(u)/d \rceil$, and this is done as in the first case.

Modifying the procedure Odd-Cycle. The procedure Odd-Cycle$'$ is the same as Odd-Cycle in terms of the performance of random walks, which are emulated as described above. The only modification is in the last stage, where the procedure performs vertex-pair queries. Let (x,y) be the pair of vertices queried in G'. We answer the query as follows. If $(x,y) = (X_i(v), I_j(v))$ for some vertex v in G, then we answer according to the explicit construction of the subgraph $H(v)$. If $(x,y) = (X_i(v), I_j(u))$ for $u \neq v$, then the answer is always negative. If $(x,y) = (X_i(u), X_j(v))$, then we query the pair (u,v) in G. If there is no edge between (u,v) in G, we answer that there is no edge between $X_i(u)$ and $X_j(v)$. Otherwise, we give a positive answer. While this answer may be inconsistent with the construction of G' (since it would correspond to having a complete bipartite subgraph in G' between the external

vertices of u and the external vertices of v), it always provides evidence to an odd cycle in the input graph G. An explanation follows.

Consider two paths in G', where both paths start at the same vertex $x \in H(s)$ and end at a pair of external vertices $X_i(v)$ and $X_j(u)$, respectively, and whose lengths have the same parity (so that $X_i(v)$ and $X_j(u)$ both belong to the same A_b, $b \in \{0,1\}$). By construction of G', such a pair of paths in G' corresponds to a pair of paths in G, which start at s, end at v and u, respectively, and have the same parity b as well. But if there is an edge in G between v and u, then there is an odd cycle in G.

4.3. Establishing item 4 in Theorem 5. In this subsection we prove the last item in Theorem 5. Recall that we are interested in a procedure for selecting a vertex in G' so that there is a sufficiently high probability of hitting any fixed sufficiently large subset of vertices in G'. In particular, if G' is far from being bipartite, then we are interested in hitting the subset of vertices s for which Odd-Cycle(s) returns found with probability at least 2/3.

Let $V_\ell(G) = \{v \in V(G) : \deg(v) \le d\}$, and let $V_h(G) = \{v \in V(G) : \deg(v) > d\}$, where "$\ell$" stands for *low*, "$h$" stands for *high*, and, as before, $d = d_{\text{avg}}(G)$ denotes the average degree of vertices in G. We also define the corresponding sets in G': $V_\ell(G') = \{x \in V(H(v)) : v \in V_\ell(G)\}$ and $V_h(G') = \{x \in V(H(v)) : v \in V_h(G)\}$. (Recall that $H(v)$ is the subgraph in G' that corresponds to v, and $V(H(v))$ is its set of vertices.) Since $V(G') = V_\ell(G') \cup V_h(G')$, it follows that selecting a vertex uniformly in $V(G')$ can be done by first deciding whether to pick a vertex from $V_\ell(G')$ or from $V_h(G')$ with probability proportional to the size of each set (relative to $n' = |V(G')|$) and then picking a vertex uniformly from the selected set.

Recall that for every $v \in V_\ell(G)$ we have $|V(H(v))| = 1$ while for every $v \in V_h(G)$, $|V(H(v))| = 2\lceil \deg(v)/d \rceil$. Therefore, picking a vertex uniformly in $V_\ell(G')$ corresponds to picking a vertex uniformly in $V_\ell(G)$, while picking a vertex uniformly in $V_h(G')$ corresponds to picking a vertex in $V_h(G)$ with probability proportional to its degree.

Since we are not required to actually select every vertex in $V(G')$ with exactly equal probability, but rather we are required to be able to select all but $\delta n'$ of the vertices in $V(G')$ with probability at least $\Omega(1/n')$, we may perform the above steps in an approximate manner. In particular, by taking a sample of $\Theta(1/\delta^2)$ vertices in G and querying their degrees, we may obtain an estimate, denoted $\hat{\mu}(G')$, of $|V_\ell(G')|/n' = |V_\ell(G)|/n'$ such that if $|V_\ell(G')|/n' \ge \delta/2$, then $(1/8)|V_\ell(G')|/n' \le \hat{\mu}(G') \le 2(|V_\ell(G')|/n')$ (recall that $n \le n' \le 4n$). In order to uniformly select a vertex in $V_\ell(G)$ (so as to obtain a uniformly selected vertex in $V_\ell(G')$), we can simply take a sample of vertices from $V(G)$, query their degrees, and pick the first vertex in the sample that belongs to $V_\ell(G)$ if such a vertex exists. If $|V_\ell(G')|/n' \ge \delta/2$, so that $|V_\ell(G)|/n \ge \delta/2$, then a sample of size $O(1/\delta)$ suffices to ensure that with high constant probability, the sample will indeed contain a vertex in $V_\ell(G)$.

The only step that is more involved is that of selecting a vertex in $V_h(G)$ with probability proportional to its degree. Observe that selecting a vertex from all of $V(G)$ with probability proportional to its degree can be performed by uniformly selecting an *edge* in $E(G)$ and then selecting one of its two endpoints with equal probability. In the next subsection we describe and analyze a procedure that performs a certain approximation to the uniform selection of an edge in $E(G)$. In subsection 4.3.2 we return to the problem of selecting a vertex in G' almost uniformly.

4.3.1. Sampling edges almost uniformly in *G*. We consider two cases: $d > \sqrt{\delta n}$ and $d \le \sqrt{\delta n}$. Recall that our goal is to use $\tilde{O}(\min(\sqrt{n/\delta}, n/d))$ queries

to G. The first case is easy since if G contains sufficiently many edges, then we simply sample $\Theta(n/d) = \Theta(n^2/m)$ pairs of vertices in order to obtain an edge.

Sample-Edges-Uniformly-in-G

Repeat $\Theta(n/d)$ times:

- Select two vertices in G uniformly and at random.
- Check if there is an edge between these two vertices (by performing a single vertex-pair query). If the answer is positive, then output this edge (and exit the repeat loop).

FIG. 4. *A procedure for sampling edges uniformly in G.*

The straightforward procedure for this case is given in Figure 4. Clearly every edge in G has equal probability of being selected by the procedure, and the query complexity of this procedure is $\Theta(n/d)$, which for $d > \sqrt{\delta n}$ is $\Theta(\min(\sqrt{n/\delta}, n/d))$ as required. (To be more precise, there is some probability that this procedure fails to output an edge. However, the probability that this occurs can be made sufficiently small so as to have a negligible effect on the success probability of our algorithm.)

In the second case, where G contains fewer edges ($d \leq \sqrt{\delta n}$), we do not have an algorithm that selects an edge uniformly from G (using relatively few queries). However, we can show the following.

LEMMA 6. *There exists a procedure* Sample-Edges-Almost-Uniformly-in-G *that uses $\tilde{O}(\sqrt{n/\delta})$ degree and neighbor queries in G and for which the following holds: For all but $(\delta/4)m$ of the edges e in G, the probability that the procedure outputs e is at least $1/(64m)$. Furthermore, there exists a subset $U_0 \subset V(G)$, $|U_0| \leq (\delta n/2)$, such that for all edges $e = (u, v)$ that are output with probability less than $1/(64m)$, we have $u, v \in U_0$.*

Sample-Edges-Almost-Uniformly-in-$G(\delta)$

1. Let $t = 2\sqrt{n/\delta} \cdot \log m$. Uniformly select a subset of vertices $S \subset V(G)$, where $|S| = t$.
2. Partition the sampled vertices into subsets according to their degree: $S_i = \{v \in S : \deg(v) \in (2^{i-1}, 2^i]\}$.
3. Choose an index i, $1 \leq i \leq \log m$, with probability $\frac{|S_i|2^i}{\sum_i |S_i|2^i}$.
4. Uniformly select a vertex $v \in S_i$.
5. Uniformly select an edge incident to v.

FIG. 5. *A procedure for selecting an edge in G so that all but at most a $(\delta/4)$-fraction of the edges are selected with probability $\Omega(1/m)$.*

The procedure referred to in Lemma 6 is described in Figure 5. Before proving Lemma 6 we provide some intuition concerning this procedure. We define the following $\log m$ "buckets": for $1 \leq i \leq \log m$,

$$B_i = \left\{v \in V(G) \ : \ \deg(v) \in (2^{i-1}, 2^i]\right\}. \tag{16}$$

Thus, in each bucket, all vertices have approximately the same degree. Suppose we had a way to pick an index i with probability proportional to $|B_i| \cdot 2^i$, which is approximately the same as picking i with probability proportional to the number of edges incident to vertices in B_i. Further assume that for each i we could uniformly

select a vertex in B_i. Then we could select an edge almost uniformly by selecting an index i as described above, uniformly selecting a vertex $v \in B_i$, and then uniformly selecting an edge incident to v.

The procedure Sample-Edges-Almost-Uniformly-in-G can be viewed as approximating this "ideal" procedure. Assume first that all buckets are relatively large (i.e., $|B_i| = \Omega(\sqrt{n})$ for every $1 \leq i \leq \log m$). Then, by taking a uniformly selected sample of $\tilde{\Theta}(\sqrt{n})$ vertices in G, we can obtain a good estimate of $|B_i|$ (and by that, of $|B_i| \cdot 2^i$), for every i, and we can uniformly select a vertex $v \in B_i$ for any given i. Unfortunately, if some buckets are small, then we might not sample from them at all.

To illustrate the seeming difficulty with such a case, suppose the graph is a star, so that there is one vertex, denoted v^* with degree $n-1$, and all other vertices have degree 1. In terms of our buckets we have $|B_0| = n-1$ so that $|B_0| \cdot 2^0 = n-1$, and $|B_{\log n}| = 1$ so that $|B_{\log n}| \cdot 2^{\log n} = n$. The "ideal" procedure would select each of the two buckets roughly with equal probability, and if it selects the bucket $B_{\log n}$, then it picks v^*. In other words, it picks v^* with probability roughly $1/2$. But if we take a sample of $\tilde{\Theta}(\sqrt{n})$ vertices, then the probability that v^* falls in the sample is extremely small. However, this turns out to be almost immaterial to the analysis since we are interested in the end result of the distribution on *edges*. In the case of the star graph, every edge incident to v^* is also incident to one of the degree-1 vertices and hence will be selected with equal probability.

In general, as we shall see in detail in the proof of Lemma 6, we can lower-bound the probability of selecting each edge that has both endpoints in sufficiently large buckets. On the other hand, we can upper-bound the total number of edges that have both endpoints in small buckets. Details follow.

Proof of Lemma 6. Let the subsets ("buckets") B_i be as defined in (16). Note that in the procedure Sample-Edges-Almost-Uniformly-in-G (Figure 5), the subsamples S_i are simply $S_i = S \cap B_i$. Next we define a set of indices I_0 that includes indices of all buckets B_i that have "few" elements. More precisely,

$$I_0 = \left\{ i \ : \ 1 \leq i \leq \log m \text{ and } |B_i| \leq \frac{\sqrt{\delta n}}{2 \log m} \right\}. \tag{17}$$

Let U_0 be the set of vertices that belong to buckets with "few" elements: $U_0 = \bigcup_{i \in I_0} B_i$.

Consider any fixed vertex $v \notin U_0$, and let $i(v)$ be the index of the bucket that v belongs to, that is, $v \in B_{i(v)}$. We denote by C_v the event that v is selected in step 4 of the sampling algorithm. Then, for a vector $(s_1, \ldots, s_{\log m})$ we can estimate

$$\Pr[C_v \ : \ |S_1| = s_1, \ldots, |S_{\log m}| = s_{\log m}] = \frac{s_{i(v)}}{|B_{i(v)}|} \cdot \frac{s_{i(v)} 2^{i(v)}}{\sum_i s_i 2^i} \cdot \frac{1}{s_{i(v)}} \tag{18}$$

(first require that v falls in $S_{i(v)}$, then choose the bucket $S_{i(v)}$, and then choose v inside $S_{i(v)}$). Thus the above conditional probability is

$$\frac{s_{i(v)}}{|B_{i(v)}|} \cdot \frac{2^{i(v)}}{\sum_i s_i 2^i} \geq \frac{s_{i(v)} \deg(v)}{|B_{i(v)}| \sum_i s_i 2^i}. \tag{19}$$

The random variable $s_{i(v)}$ is hypergeometrically distributed with parameters n, $|B_{i(v)}|$, and t. It thus has mean $t|B_{i(v)}|/n$, and using known bounds on the tails of the

hypergeometric distribution and our assumption on v ($v \notin U_0$ and therefore $B_{i(v)}$ is large), we can get

$$\Pr\left[\frac{s_{i(v)}}{|B_{i(v)}|} \geq \frac{t}{2n}\right] \geq \frac{3}{4}.$$

Consider the sum $\sum_{i=1}^{t} s_i 2^i$. We have

$$\begin{aligned}\mathrm{Exp}\left[\sum_i |S_i|2^i\right] &= \sum_i \frac{|B_i|}{n} \cdot t \cdot 2^i \\ &= \frac{t}{n} \cdot \sum_i |B_i|2^i \\ &\leq \frac{t}{n} \cdot \sum_i \sum_{v \in B_i} (2\deg(v)) \quad \leq \quad \frac{4 \cdot t \cdot m}{n}.\end{aligned}$$

By Markov's inequality, the probability that $\sum_i |S_i|2^i > \frac{16tm}{n}$ is less than $1/4$. It thus follows that

$$\Pr\left[\left(\frac{s_{i(v)}}{|B_{i(v)}|} \geq \frac{t}{2n}\right) \& \left(\sum_i s_i 2^i \leq \frac{16tm}{n}\right)\right] \geq \frac{1}{2}. \tag{20}$$

Consider only such vectors $(s_1, \ldots, s_t)$. From (19) we obtain

$$\Pr[C_v] \geq \frac{1}{2} \cdot \frac{t}{2n} \cdot \frac{\deg(v)}{\frac{16tm}{n}} = \frac{\deg(v)}{64m}. \tag{21}$$

Finally, for an edge $e \in E(G)$, let us denote by C_e the event that e is selected in the fourth step of the procedure Sample-Edges-Almost-Uniformly-in-G. Let $E(U_0)$ denote the set of edges between pairs of vertices in U_0, that is, $E(U_0) = \{(u,v) \in E(G) : u, v \in U_0\}$. By definition of U_0, $|U_0| \leq \sqrt{\delta n}/2$, and hence $|E(U_0)| \leq (\delta n)/4$. Therefore, for all but at most $(\delta n)/4 \leq \delta m/4$ of the edges in $E(G)$, at least one of their endpoints is *not* in U_0. (Note that here we have used the assumption that $m \geq n$.) For each such edge (u,v) where $u \notin U_0$ (or $v \notin U_0$), we have

$$\Pr[C_e] \geq \Pr[C_u] \cdot \frac{1}{\deg(u)} \geq \frac{\deg(u)}{64m} \cdot \frac{1}{\deg(u)} = \frac{1}{64m}. \tag{22}$$

□

4.3.2. Sampling vertices in G'. We now return to selecting vertices in G'. As discussed earlier, we can easily obtain an estimate of $|V_\ell(G')|/n' = |V_\ell(G)|/n'$, denoted $\hat{\mu} = \hat{\mu}(G')$, such that if $|V_\ell(G')|/n' \geq \delta/2$, then $(1/8)|V_\ell(G')|/n' \leq \hat{\mu}(G') \leq 2(|V_\ell(G')|/n')$, and hence we assume that we indeed have such an estimate.

Proof of item 4 *in Theorem* 5. We now show that the procedure Sample-Vertices-Almost-Uniformly-in-G' (see Figure 6) is as required in item 4 of Theorem 5. Consider first the vertices in $V_\ell(G')$. If $|V_\ell(G')|/n' \geq \delta/2$, then for each vertex $x \in V_\ell(G')$, the probability that we select x is $\hat{\mu}(G')$, times the probability that the sample contains a vertex in $V_\ell(G')$, times $1/|V_\ell(G')|$. Since $\hat{\gamma} = \Omega(|V_\ell(G')/n')$ and the sample contains a vertex from $V_\ell(G)$ with constant probability, the probability that we select x is $\Omega(1/n')$ as required. If $|V_\ell(G')|/n' < \delta/2$, then the probability that we obtain any vertex in $V_\ell(G')$ may be very small, but we are allowed to have $\delta n'$ such vertices and we shall account for these at most $(\delta/2)n'$ vertices.

Sample-Vertices-Almost-Uniformly-in-$G'(d, \delta, \hat{\mu})$

1. Flip a coin with bias $\hat{\mu}$.
2. If the outcome is "heads," then do (select a vertex in $V_\ell(G')$):
   (a) Uniformly select $\Theta(1/\delta)$ vertices in G and query their degrees.
   (b) If some vertex in the sample belongs to $V_\ell(G)$, then let v be the first such vertex and output the single vertex $x \in V(H(v))$. Otherwise, pick an arbitrary vertex v in the sample and output an arbitrary vertex $x \in V(H(v))$.
3. Else (the outcome is "tails"), do (select a vertex in $V_h(G')$):
   (a) If $d > \sqrt{\delta n}$, then sample an edge $e \in E(G)$ by running the procedure Sample-Edges-Uniformly-in-G. In case the procedure fails, pick an arbitrary edge e in $E(G)$.
   (b) Else ($d \leq \sqrt{\delta n}$), sample an edge $e \in E(G)$ by running the procedure Sample-Edges-Almost-Uniformly-in-$G(\delta/2)$.
   (c) Choose with equal probability one of the endpoints v of the edge e.
   (d) Choose uniformly at random one of the vertices x in $V(H(v))$.

FIG. 6. *A procedure for selecting a vertex in G' so that all but at most a δ-fraction of the vertices are selected with probability $\Omega(1/n')$.*

We now turn to the vertices in $V_h(G')$. For an edge $e \in E(G)$, let C_e denote the event that e is selected by Sample-Edges-Uniformly-in-G in case $d > \sqrt{\delta n}$, or by Sample-Edges-Almost-Uniformly-in-G in case $d \leq \sqrt{\delta n}$. For $v \in V(G)$ let C_v denote the event that v is selected in step 3(c) of procedure Sample-Vertices-Almost-Uniformly-in-G'. For $x \in V(G')$ let C_x denote the event that x is selected in step 3(d) of the procedure, and let $v(x)$ be such that $x \in H(v(x))$. Recall that for every $v \in V_h(G)$, we have $|V(H(v))| = 2\lceil \deg(v)/d \rceil$. Therefore, for every $x \in V_h(G')$,

$$\Pr[C_x] \geq \Pr[C_{v(x)}] \cdot \frac{1}{2\lceil \deg(v(x))/d \rceil} = \sum_{e=(u,v(x))} \frac{1}{2} \Pr[C_e] \cdot \frac{1}{2\lceil \deg(v(x))/d \rceil}. \tag{23}$$

If $d > \sqrt{\delta n}$, then $\Pr[C_e]$ is only slightly smaller than $1/m$ (since there is a probability that the procedure Sample-Edges-Uniformly-in-G fails to output an edge), implying that $\Pr[C_x] = \Omega(1/n')$. If $d \leq \sqrt{\delta n}$, then $\Pr[C_e]$ is determined by the procedure Sample-Edges-Almost-Uniformly-in-G. Let U_0 be as defined in Lemma 6, and consider first the case where $v(x) \notin U_0$. Then for every edge e that is incident to $v(x)$, we have that $\Pr[C_e] \geq 1/(64m) = 1/(32dn)$. By (23), for each such vertex x we have that

$$\Pr[C_x] \geq \frac{1}{2}\deg(v(x)) \cdot \frac{1}{32dn} \cdot \frac{1}{2\lceil \deg(v(x))/d \rceil} \geq \frac{1}{256n} \geq \frac{1}{256n'} \tag{24}$$

as required. Next consider the case that $v(x) \in U_0$ but $\deg(v(x)) \geq 2|U_0|$. In such a case for at least half of the edges e incident to $v(x)$ we have that $\Pr[C_e] \geq 1/(32dn)$, and we can deduce that $\Pr[C_x] \geq \frac{1}{512n'}$. It remains to show that the total number of vertices x such that $v(x) \in U_0$ and $\deg(v) \leq 2|U_0|$ is at most $(\delta/2)n'$. Using the fact that $|U_0| \leq \sqrt{(\delta/2)n}/2$ (recall that the procedure Sample-Edges-Almost-Uniformly-in-G is called with its input parameter set to $\delta/2$), and for every vertex $v \in V_h(G)$, $|V(H(v))| = 2\lceil \deg(v)/d \rceil \leq 2\deg(v)$, we get

$$\sum_{v \in U_0:\ \deg(v) \leq 2|U_0|} |V(H(v))| \leq 4|U_0|^2 \leq (\delta/2)n'. \tag{25}$$

The lemma follows.

4.4. A probabilistic construction of G'. In this subsection we describe an alternative, probabilistic construction of a graph G' that establishes Theorem 5. More precisely, we describe such a construction that, under certain conditions on d and ϵ, results, with very high probability, in a graph G' as required in the theorem. Here too the graph is constructed as the algorithm proceeds. In all that follows, let $d = d_{\text{avg}}(G)$, and let $d' = d_{\max}(G')$. Let n (n') be the number of vertices in G (G'), and m (m') be the number of edges in G (G'). The probabilistic construction is simpler, and it is possible that it may be applicable to other problems as well. However, in this construction we need that $d = \Omega(\log n + 1/\epsilon)$.

4.4.1. The construction. As in the deterministic construction, every vertex of G is transformed into $\lceil \deg(v)/d \rceil$ vertices. Denote by $X(v)$ the vertices in G' related to a vertex $v \in V(G)$. The vertices in $X(v)$ are denoted by $X_i(v), 1 \le i \le \lceil \deg(v)/d \rceil$. Thus, $n' = |V(G')| \le \sum_{v \in G} \lceil \frac{\deg(v)}{d} \rceil \le 2n$. The edges of G' are determined as follows: an edge $(u, v) \in E(G)$ chooses independently uniformly at random a vertex from $X(v)$ and a vertex from $X(u)$. In G' there will be an edge between these two randomly chosen vertices. Clearly, $m' = |E(G')| = |E(G)| = (nd)/2$.

LEMMA 7. *For $d = \Omega(\log n)$, the maximum degree d' of G' constructed above is $2d$ with probability $1 - o(1)$.*

Proof. Let $W_v^{i,j}$ be an indicator random variable which is 1 if the jth induced edge of $v \in V(G)$ ($1 \le j \le \deg(v)$) chooses vertex $X_i(v) \in V(G')$. Let $W_v^i \stackrel{\text{def}}{=} \sum_{1 \le j \le \deg(v)} W_v^{i,j}$. W_v^i is a sum of j independent indicator variables. $\text{Exp}[W_v^i] = \deg(v) \cdot 1/(\deg(v)/d) = d$. Let X_v^i be an indicator variable which is 1 if $W_v^i > 2\text{Exp}[W_v^i] = 2d$ and 0 otherwise.

Using standard bounds on tails of sums of bounded random variables (see, e.g., [5]) for a specific $v \in V(G)$ and $1 \le i \le \lceil \deg(v)/d \rceil$, it follows that $\Pr[X_v^i = 1] < e^{-cd}$. Using a union bound over all X_v^i's, $v \in V(G)$ and $1 \le i \le \lceil \deg(v)/d \rceil$, we get that the probability that there exist a vertex v and an index i for which $X_v^i = 1$ is at most $n' \cdot e^{-cd} = o(1)$; thus with probability $1 - o(1)$, the maximum degree of G' constructed above is $2d$. ∎

LEMMA 8. *For a graph G with $d > 512/\epsilon$ the following holds: If G is ϵ-far from being bipartite (with respect to $m = (dn)/2$), then with probability $1 - o(1)$, G' is ϵ'-far from being bipartite with respect to $d'n'$ for $\epsilon' = \frac{\epsilon}{128}$.*

Proof. Consider a fixed partition $P' = (V_0', V_1')$ of the vertices in G'. The partition P' induces a partition of $X(v)$ for every v. Let us denote by $X^\alpha(v)$ the majority subset of $X(v)$ induced by P'. Consider a partition $P = (V_0, V_1)$ of the vertices of G induced by P' in the following way. For $v \in V(G)$, if $X^\alpha(v) \subset V_0'$, then $v \in V_0$; otherwise $v \in V_1$. Since G is ϵ-far from being bipartite, at least one of the subsets V_0, V_1 contains $\frac{1}{4}\epsilon nd$ edges. Without loss of generality, assume that $|E(V_0)| \ge \frac{1}{4}\epsilon nd$. Let H' be a subgraph of G', defined as follows. The vertices of H' are

$$\bigcup_{v \in V_0} X^\alpha(v).$$

The edges of H' are the edges of G', induced by $V(H')$. Thus, $V(H') \subset V_0'$ and $E(H') \subset E(V_0')$.

CLAIM 2. $\Pr[|E(H')| \le (\epsilon/32)nd] < 2^{-c \cdot n}$, *where $c > 1$.*

Once Claim 2 is proved, by taking a union bound over all possible partitions P' of the vertices of G' we get that for *every* partition $P' = (V_0', V_1')$ of the vertices of G',

the number of violating edges in G' is at least $\epsilon/32 \cdot n \cdot d$ with probability $1 - o(1)$. Recall that $n' \leq 2n$ and $d' \leq 2d$ with probability $1 - o(1)$. Thus, for *every* partition of the vertices of G', the number of violating edges is at least $\epsilon/128 \cdot n' \cdot d'$ with probability $1 - o(1)$, as required.

Proof of Claim 2. Consider an edge $e = (u, v) \in E(G)$, and let $\phi(e) = (X_i(v), X_j(u))$ be its corresponding edge in $E(G')$. For $e \in E(V_0)$, $\Pr[\phi(e) \in E(H')] \geq 1/2 \cdot 1/2 = 1/4$. Thus,

$$\mathrm{Exp}[|E(H')|] \geq 1/4|E(V_0)|.$$

As $|E(H')|$ is a sum of $|E(V_0)|$ independent Bernoulli random variables, each with expectation at least $1/4$, it follows from standard bounds on the tails of binomial random variables (see, e.g., [5]) that

$$\Pr[X_{|E(V_0)|} < (1/32)\epsilon nd] < e^{-\epsilon nd/512}.$$

Thus, for $d \geq 512/\epsilon$ we have

$$\Pr[|E(H')| \leq \epsilon/32nd] < 2^{-cn}$$

for $c > 1$, and the lemma is proved. □

We have completed proving the first two items of Theorem 5, with regard to the probabilistic construction.

4.4.2. Proving item 3 in Theorem 5. In order to prove this item we need to explain how to emulate queries in G' by performing queries in G.

Degree queries. Here we assume that in G' the maximum degree is $2d$. It was proved before to be true with probability $1 - o(1)$. Consider the following procedure Assign-Edges(v), where v is a vertex of G: Find $\deg(v)$ by one query on G; for each edge-index $r \in \{1, \ldots, \deg(v)\}$, choose one of the vertices $X_i(v) \in X(v)$ ($1 \leq i \leq \lceil \deg(v)/d \rceil$) uniformly at random. Denote by $Q_i(v)$ a vector of $2d$ cells. This vector contains the indices of edges that are assigned to $X_i(v)$. Finally, choose an empty cell t in the vector $Q_i(v)$, and set $Q_i(v)[t] := r$. This procedure corresponds to the random construction of G'.

Neighbor queries. For each vertex $X_i(v)$, the algorithm keeps a vector (of length $2d$), $W_i(v)$ that contains an ordered list of all the vertices of G' such that the algorithm is committed to the existence of an edge between them and $X_i(v)$. That is, if $W_i(v)[t] = X_j(u)$, it means that there is an edge $(X_i(v), X_j(u))$ in G'. In this case $W_j(u)[t'] = X_i(v)$ for some t', $1 \leq t' \leq 2d$. Note that the procedure Assign-Edges(v) assigns each of the edges of v independently to one of the vertices $X_i(v)$, $1 \leq i \leq \lceil \deg(v)/d \rceil$.

Suppose that the algorithm is located in $X_i(v)$ and it wishes to perform a neighbor query. Choose uniformly at random a number k, $1 \leq k \leq 2d$. Take an empty vector $Q_i(v)$ of length $2d$ and copy into it the vector $W_i(v)$. By that the algorithm keeps its commitment about the edges which have already been exposed. Then, apply the Assign-Edges(v) procedure only for edges of v that do not appear in one of the $W_i(v)$, $1 \leq i \leq \lceil \deg(v)/d \rceil$ (i.e., to edges which were not exposed yet by the algorithm). If $Q_i(v)[k]$ is empty, then the walk remains at $X_i(v)$. If $Q_i(v)[k]$ contains a name of a vertex in G', the walk moves to that vertex; otherwise, $Q_i(v)[k]$ contains an index of a neighbor of v in G. By performing a single neighbor query on G, the algorithm determines the name of that neighbor (say) u in G. Then the walk needs to move to one of the vertices $X_j(u)$, $1 \leq j \leq \deg(u)/d$. It chooses one of the vertices $X_j(u)$

uniformly at random. For the chosen j set $W_i(v)[k] := X_j(u)$. Now choose a free cell k', $1 \le k' \le 2d$, in $W_j(v)$, and set $W_j(v)[k'] := X_i(v)$. The walk is now at $X_j(u)$.

Modifying the procedure Odd-Cycle. The procedure is modified in a similar manner to that described for the deterministic construction. In particular, random walks are emulated as described above. In the last stage, when vertex-pair queries are performed, we do the following. In order to answer a vertex-pair query $(X_i(v), X_j(u))$ in G', perform a vertex-pair query (u, v) in G. If there is no edge between (u, v) in G, answer that there is no edge between $X_i(u)$ and $X_j(v)$. Otherwise, as explained for the deterministic case, an odd cycle is detected in G.

4.4.3. Establishing item 4 in Theorem 5. In this subsection we prove the last item in Theorem 5 (for the probabilistic construction). The proof is essentially a simplified version of the proof for the deterministic case, but for the sake of completeness we include the details.

Recall that we are interested in a procedure for selecting a vertex in G' such that there is sufficiently high probability of hitting any fixed sufficiently large subset of vertices in G' (and, in particular, of hitting the subset of vertices s for which Odd-Cycle(s) returns found with probability at least 2/3 (in the case G' is far from being bipartite)).

Sample-Vertices-Almost-Uniformly-in-$G'(d, \delta)$

1. If $d > \sqrt{\delta n}$, then sample an edge $e \in E(G)$ by running the procedure Sample-Edges-Uniformly-in-G. In case the procedure fails, pick an arbitrary edge e in $E(G)$.
2. Else ($d \le \sqrt{\delta n}$), sample an edge $e \in E(G)$ by running the procedure Sample-Edges-Almost-Uniformly-in-$G(\delta/2)$.
3. Choose with equal probability one of the endpoints v of the edge e.
4. Choose uniformly at random one of the vertices x in $X(v)$.

FIG. 7. *A procedure for selecting a vertex in G' so that all but at most a δ-fraction of the vertices are selected with probability $\Omega(1/n')$.*

We now show that the procedure Sample-Vertices-Almost-Uniformly-in-G' (see Figure 7) is as required in item 4 of Theorem 5.

For an edge $e \in E(G)$, let C_e denote the event that e is selected by Sample-Edges-Uniformly-in-G in case $d > \sqrt{\delta n}$, or by Sample-Edges-Almost-Uniformly-in-G in case $d \le \sqrt{\delta n}$. For $v \in V(G)$ let C_v denote the event that v is selected in step 3 of procedure Sample-Vertices-Almost-Uniformly-in-G'. For $x \in V(G')$ let C_x denote the event that x is selected in step 4 of the procedure, and let $v(x)$ be such that $x \in X(x(v))$. Recall that $|X(v)| = \lceil \deg(v)/d \rceil$. Therefore, for every $x \in V(G')$,

$$\Pr[C_x] \;\ge\; \Pr[C_{v(x)}] \cdot \frac{1}{\lceil \deg(v(x))/d \rceil} \;=\; \sum_{e=(u,v(x))} \frac{1}{2} \Pr[C_e] \cdot \frac{1}{\lceil \deg(v(x))/d \rceil}. \tag{26}$$

If $d > \sqrt{\delta n}$, then $\Pr[C_e]$ is only slightly smaller than $1/m$ (since there is a probability that the procedure Sample-Edges-Uniformly-in-G fails to output an edge), implying that $\Pr[C_x] = \Omega(1/n')$. If $d \le \sqrt{\delta n}$, then $\Pr[C_e]$ is determined by the procedure Sample-Edges-Almost-Uniformly-in-G. Let U_0 be as defined in Lemma 6, and consider first the case where $v(x) \notin U_0$. Then for every edge e that is incident to $v(x)$, we have that $\Pr[C_e] \ge 1/(64m) = 1/(32dn)$. By (26), for each such vertex x we have that

$$\Pr[C_x] \geq \frac{1}{2}\deg(v(x)) \cdot \frac{1}{32dn} \cdot \frac{1}{\lceil \deg(v(x))/d \rceil} \geq \frac{1}{128n} \geq \frac{1}{128n'} \tag{27}$$

as required. Next consider the case that $v(x) \in U_0$ but $\deg(v(x)) \geq 2|U_0|$. In such a case for at least half of the edges e incident to $v(x)$ we have that $\Pr[C_e] \geq 1/(32dn)$, and we can deduce that $\Pr[C_x] \geq \frac{1}{256n'}$. It remains to show that the total number of vertices x such that $v(x) \in U_0$ and $\deg(v) \leq 2|U_0|$ is at most $(\delta/2)n'$. Using the fact that $|U_0| \leq \sqrt{(\delta/2)n}/2$ (recall that the procedure Sample-Edges-Almost-Uniformly-in-G is called with its input parameter set to $\delta/2$), and for every vertex $v \in V(G)$, $|X(v)| = \lceil \deg(v)/d \rceil \leq \deg(v)$, we get

$$\sum_{v \in U_0:\ \deg(v) \leq 2|U_0|} |X(v)| \ \leq \ 2|U_0|^2 \ \leq \ (\delta/2)n'. \tag{28}$$

Thus, item 4 of Theorem 5 follows.

5. A lower bound. In this section we present a lower bound on the number of queries necessary for testing bipartiteness. Similarly to the lower bound presented in [14], this lower bound holds for testing algorithms that are allowed a two-sided error, and the graphs used for the lower bound construction are regular graphs. However, the lower bound of $\Omega(\sqrt{n})$ (for constant ϵ) established in [14] holds for graphs having constant degree (e.g., degree 3), and when the algorithm is allowed only neighbor queries. Our lower bound is more general in that it allows the algorithm to perform both neighbor queries and vertex-pair queries, and it is applicable to all degrees. Indeed, the two families of graphs that we define below in our lower bound construction can be viewed as generalizing the two families presented in [14]. However, since we have to deal with any given degree d and not only with $d = 3$, and since we have to deal with both types of queries, the analysis itself does not follow as a straightforward generalization of the analysis in [14].

THEOREM 8. *Every algorithm for testing bipartiteness with distance parameter* $\epsilon \leq 2^{-4}$ *must perform* $\Omega(\min(\sqrt{n}, n^2/m))$ *queries.*

The high level structure of our proof is similar to other lower-bound proofs for testing, which can be traced back to [22]. Specifically, we present two distributions over graphs (which are defined in detail below): $\mathcal{G}(n,d)$ and $\mathcal{G}(n/2, n/2, d)$, where d is the degree of the vertices in the graphs. (For simplicity we assume that n is even.) We prove that graphs created randomly according to $\mathcal{G}(n,d)$ are ϵ-far from being bipartite with high probability, while all graphs in the support of $\mathcal{G}(n/2, n/2, d)$ are bipartite. We then show that a bipartite tester that asks $o(\min(\sqrt{n}, n^2/m))$ queries cannot distinguish with sufficiently high probability whether a graph is created according to the distribution $\mathcal{G}(n,d)$ or $\mathcal{G}(n/2, n/2, d)$. Note that by definition, in both cases $m = (nd)/2$, and so $n^2/m = 2n/d$.

The graph distribution $\mathcal{G}(n,d)$. A graph G in the support of the distribution $\mathcal{G}(n,d)$ is composed of n vertices, and it is a d-regular graph. The edges of G are determined according to the following random process. Consider a two-dimensional table of size $n \times d$, which we refer to as the *matching table*. Each cell in the matching table is denoted by $c_{u,i}$, where u denotes the row in which the cell is located (and corresponds to a vertex in the graph) and i denotes the column in which the cell is located (and corresponds to a label of an edge incident to v).

Consider a perfect matching over the cells of the table, randomly chosen over all possible perfect matchings. This randomly chosen matching defines the edges of the graph G. That is, if the cells $c_{u,i}$ and $c_{v,j}$ are matched, then this means that there is an edge (u, v, i, j) in the graph.

The graph distribution $\mathcal{G}(n/2, n/2, d)$. A graph G in the support of the distribution $\mathcal{G}(n/2, n/2, d)$ is a bipartite d-regular graph composed of n vertices, where each side of the partition is of size $n/2$. The edges of G are determined according to the following random process. First, we select a random partition of the graph vertices $\{1, \ldots, n\}$ into two parts (sides), each of size $n/2$. Second, consider two tables, each of size $n/2 \times d$, where each is attached to one of the partition sides. The rows in each table refer to the vertices that belong to the relevant side of the partition. Let us refer to the table attached to the first side of the partition as the *even table* and to the table attached to the second side of the partition as the *odd table*.

A cell in these tables is denoted by $c_{b,u,i}$, where $b \in \{1, 2\}$ denotes the relevant table, u denotes the row in which the cell is located, and i denotes the column in which the cell is located.

Now, consider a perfect matching over the cells of the two tables, randomly chosen over all possible perfect matchings that are restricted to matching cells from the odd table with cells from the even table. This randomly chosen matching defines the edges of the graph G. That is, if the cell $c_{1,u,i}$ is matched to the cell $c_{2,v,j}$, this means that there is an edge (u, v, i, j) in the graph (where u belongs to one side of the bipartite graph and v to the other side).

By definition, all graphs in the support of $\mathcal{G}(n/2, n/2, d)$ are bipartite. We can show that almost all graphs in the support of $\mathcal{G}(n, d)$ are far from being bipartite.

LEMMA 9. *With probability* $1 - o(1)$*, a graph chosen uniformly according to the distribution* $\mathcal{G}(n, d)$ *is* ϵ*-far from being bipartite for every* $\epsilon \leq 1/16$ *and* $d \geq 64$.

Proof. We shall show that for every fixed partition (V_0, V_1) of V, the probability, taken over the selection of a graph in $\mathcal{G}(n, d)$, that there are more than $\epsilon \cdot n \cdot d$ violating edges with respect to (V_0, V_1), is very close to one. The lemma will then follow by a union bound (over all partitions).

Let $P = (V_0, V_1)$ be an arbitrary fixed partition of V. Without loss of generality, let $|V_0| \geq n/2$. Consider the following process, denoted by C_P, for choosing a random d-regular graph having n vertices (i.e., a perfect matching over a matching table of size $n \times d$). Starting from $b = 0$, choose an arbitrary cell $c_{u,i}$ such that $u \in V_b$, and match $c_{u,i}$ to a randomly chosen unmatched cell $c_{v,j}$. If the number of unmatched cells that belong to vertices in V_b is smaller than the number of unmatched cells that belong to the other side of the partition, then switch sides (i.e., let $b \leftarrow 1 - b$). Finish when all the cells are matched. Thus, the number of steps in this process is $\frac{nd}{2} - 1$. (In the last step there are two unmatched cells, which are matched together.) We denote by G_P the graph chosen according to process C_P.

CLAIM 3. *For every fixed partition* P*, the distribution on graphs induced by the process* C_P *is identical to* $\mathcal{G}(n, d)$.

Proof. For each step t in the process C_P, $1 \leq t \leq \frac{nd}{2} - 1$, let e_t be the edge selected in that step. Let R_0 be the set of all graphs in the support of $\mathcal{G}(n, d)$, and denote by $R_t \subset R_0$ the set of graphs in R_0 that contain the edges $e_1, \ldots, e_t$. Using this notation, at the start of process C_P, before any edge is selected (any two cells are matched), the process can potentially select any graph in R_0. After performing t steps, it is restricted to selecting graphs from R_t.

The probability that a particular graph G in the support of $\mathcal{G}(n, d)$ is selected is

$$\Pr[G_P = G] \;=\; \Pr[G \in R_1] \cdot \Pr[G \in R_2 | G \in R_1] \cdot \;\cdots\; \cdot \Pr[G \in R_{\frac{nd}{2}-1} | G \in R_{\frac{nd}{2}-2}].$$

Consider any particular term $\Pr[G \in R_t | G \in R_{t-1}]$ in the above expression. Conditioned on G belonging to R_{t-1} (that is, the edges $e_1, \ldots, e_{t-1}$ selected by the

process C_P are all edges in G), we would like to know what the probability is that G belongs to R_t as well. Let $c_{u,i}$ be the arbitrary unmatched cell selected by C_P in step t. Let v be the ith neighbor of u in G, where u is the jth neighbor of v. Since $G \in R_{t-1}$, necessarily $c_{v,j}$ is an unmatched cell in the matching table. Hence, the probability that G belongs to R_t (conditioned on it belonging to R_{t-1}) equals the probability that $c_{u,i}$ is matched to $c_{v,j}$, which is $1/(dn - 2(t-1) - 1)$. Therefore,

$$\begin{aligned}\Pr[G_P = G] &= \frac{1}{dn-1} \cdot \frac{1}{dn-3} \cdot \cdots \cdot \frac{1}{3} \;=\; \frac{dn \cdot (dn-2) \cdot \cdots \cdot 2}{(dn)!} \\ &= \frac{(2(dn/2)) \cdot (2((dn/2)-1)) \cdot \cdots \cdot 2}{(dn)!} \;=\; \frac{2^{dn/2} \cdot (dn/2)!}{(dn)!}. \end{aligned} \tag{29}$$

It remains to show that the above expression equals $1/|R_0|$. But this is easy to verify. By definition, $|R_0|$ is the total number of possible perfect matchings between the dn cells in the matching table. Each such matching can be obtained by ordering all the cells and letting the matching be defined by successive pairs. The number of such orderings is $(dn)!$. However, different orderings may produce the same matching. In particular, we need to take into account the overcounting due to orderings within the pairs (i.e., the pairs $(c_{v,i}, c_{u,j})$ and $(c_{u,j}, c_{v,i})$ are the same) and the overcounting due to orderings between the different pairs. In other words, we need to divide $(dn)!$ by $2^{dn/2} \cdot (dn/2)!$. We have thus obtained the inverse of the expression in (29), as required. □

We have thus completed proving the claim and may view a graph G selected uniformly from $\mathcal{G}(n,d)$, as if it is created according to the process C_P, for any fixed partition P. It remains to show that with very high probability over the choice of such a graph, it has more than $\epsilon \cdot nd$ violating edges with respect to the partition P.

By definition, during the process C_P, we always try to match a cell that belongs to a side of the partition having more unmatched cells. Thus, at each step we create a violating edge with probability at least $\frac{1}{3}$ (except the last stage), independent of the past events. It follows that the expected number of violating edges in the final resulting graph G, with respect to a particular partition P, is $\mu \geq (nd/2 - 1) \cdot \frac{1}{3} > \frac{nd}{8}$. Let Y_P denote the number of violating edges with respect to P. By applying the Chernoff bound we have that for any constant $0 < \alpha < 1$,

$$\Pr[Y_P < (1-\alpha)\mu] < \exp(-(1/2)\alpha^2\mu).$$

In particular, for $\alpha = 1 - 8\epsilon$ (recall that $\epsilon \leq 1/16$) we obtain

$$\begin{aligned}\Pr[Y_P < \epsilon nd] &\leq \Pr[Y_P < 8\epsilon\mu] \\ &< \exp(-(1/2)(1-8\epsilon)^2\mu) \\ &\leq \exp(-(1-8\epsilon)^2 nd/16).\end{aligned}$$

Since the total number of partitions is 2^n, if we take a union bound over all possible partitions, then we can deduce that the probability that G has less than ϵnd violating edges with respect to *any* partition P is bounded from above by $2^n \cdot \exp(-(1-8\epsilon)^2 nd/16)$. Taking $\epsilon \leq 1/16$ and $d \geq 64$, we get that a graph constructed according to a distribution $\mathcal{G}(n,d)$ is ϵ-far from being bipartite with probability $1 - o(1)$ as required. □

Let ALG be an algorithm for testing bipartiteness using $Q = Q(n)$ queries. Namely, ALG is a (possibly probabilistic) mapping from *query-answer histories*

$\langle (q_1, a_1), \cdot, (q_t, a_t) \rangle$ to q_{t+1} for every $t < Q$, and to $\{\mathsf{accept}, \mathsf{reject}\}$ for $t = Q$. Recall that a query q_t can be of two types: a neighbor query and a vertex-pair query. A neighbor query q_t is a pair (u_t, i_t), where $u_t \in V$ and $i_t \in \{1, \ldots, \deg(u_t)\}$. Here the corresponding answer a_t is a pair (v_t, j_t), where $v_t \in V$ and $j_t \in \{1, \ldots, \deg(v_t)\}$ such that v_t is the i_tth neighbor of u_t and u_t is the j_tth neighbor of v_t. A vertex-pair query q_t is a pair (u_t, v_t), where $u_t, v_t \in V$, and the corresponding answer is of the form $a_t = (y_t, i_t, j_t)$. If there is an edge between u_t and v_t, then $y_t = 1$, and i_t and j_t are the corresponding labels of the edge. If no such edge exists, then $y_t = 0$, and i_t and j_t are set arbitrarily, say, to 1. In the former case we shall say that ALG has *detected an edge* (by a vertex-pair query). In the latter case we shall say that (u_t, v_t) is a *nonedge* (and that ALG has detected a nonedge).[5] Note that the answers to the queries of ALG contain additional indexing information that does not appear in our model. We show that even by using the extra information, ALG cannot do better than the stated lower bound.

We assume that the mapping determined by the algorithm is defined only on histories that are consistent with some n-vertex graph. Any query-answer history of length t can be used to define a *knowledge graph* G_t at time t. The vertex set of G_t contains all vertices that appear in the history (either in queries or in answers). For every neighbor query $(u_{t'}, i_{t'})$ answered by $(v_{t'}, j_{t'})$ $(t' \leq t)$, the graph G_t contains the edge $(u_{t'}, v_{t'}, i_{t'}, j_{t'})$, and similarly for every vertex-pair query $(v_{t'}, u_{t'})$ that is answered by $(1, i_{t'}, j_{t'})$. In addition, for every vertex-pair query $(v_{t'}, u_{t'})$ that is answered by $(0, 1, 1)$, the knowledge graph maintains the information that $(v_{t'}, u_{t'})$ is a nonedge. Thus G_t is a subgraph of the graph tested by ALG.

In what follows we describe two random processes, P^1 and P^2, which interact with an arbitrary algorithm ALG. The process P^1 answers ALG's queries while constructing a random graph from $\mathcal{G}(n, d)$, and the process P^2 answers ALG's queries while constructing a random graph from $\mathcal{G}(n/2, n/2, d)$. We assume without loss of generality that ALG does not ask queries whose answer can be derived from its knowledge graph, since such queries give it no new information. (For example, ALG does not ask a vertex-pair query about a pair of vertices which are already known to be connected by an edge due to a neighbor query.)

For a fixed algorithm ALG that uses Q queries, and for $b \in \{1, 2\}$, let D^b_{ALG} denote the distribution on query-answer histories (of length Q) induced by the interaction of ALG and P^b. We show that for any given ALG that uses $o(\min(\sqrt{n}, n/d))$ queries, the statistical distance between D^1_{ALG} and D^2_{ALG} is $o(1)$. Combining this with Lemma 9 (and the fact that $m = (nd)/2$), Theorem 8 follows.

Definition of P^b, $b \in 1, 2$.

- Let R^1 be the set of all graphs in the support of $\mathcal{G}(n, d)$, and let R^2 be the set of all graphs in the support of $\mathcal{G}(n/2, n/2, d)$.
  For an edge $e_k = (u, v, i, j)$, denote by $R^b_{e_k} \subset R^b$ the subset of graphs in R^b that contain e_k. We may refer to $R^b_{e_k}$ as $R^b_{(u,v,i,j)}$.
  For an edge-pair $f_k = (u, v)$, denote by $R^b_{f_k} \subset R^b$ the subset of graphs in R^b that contain an edge between the vertices u, v. Denote by $R^b_{\overline{f_k}} \subset R^b$ the subset of graphs in R^b that do not contain an edge between the vertices u, v.
- The process P^b answers queries as follows. Initialize $R^b_0 = R^b$ (in general, as is explained in more detail below, for any $t \geq 0$, the set R^b_t consists of all graphs in R^b that are consistent with the first t queries and answers).

[5] In case multiple edges are allowed, then the answer is of the form (y_t, I_t, J_t), where I_t and J_t are sets of labels.

1. If the tth query is a vertex-pair query $q_t = (u_t, v_t)$, then the answer $a_t = (y_t, i_t, j_t)$ is selected as follows. Let $y_t = 1$ with probability $|R^b_{(u_t,v_t)} \cap R^b_{t-1}|/|R^b_{t-1}|$, and let $y_t = 0$ with probability $1 - |R^b_{(u_t,v_t)} \cap R^b_{t-1}|/|R^b_{t-1}|$. In the former case, for any pair $i, j \in \{1, \ldots, d\}$, the probability that $i_t = i$ and $j_t = j$ is $|R^b_{(u_t,v_t,i,j)} \cap R^b_{t-1}|/|R^b_{t-1}|$, and we set $R^b_t = R^b_{(u_t,v_t,i,j)} \cap R^b_{t-1}$. In the latter case, the values of i_t and j_t can be selected arbitrarily (say, to 1), and we set $R^b_t = R^b_{\overline{(v_t,u_t)}} \cap R^b_{t-1}$.
2. If the tth query is a neighbor query $q_t = (u_t, i_t)$, then for each $v \in V$, $j \in \{1, \ldots, d\}$, answer $a_t = (v, j)$ with probability $|R^b_{(u_t,v,i_t,j)} \cap R^b_{t-1}|/|R^b_{t-1}|$, and set $R^b_t = R^b_{(u_t,v_t,i,j)} \cap R^b_{t-1}$.

- After all queries are answered (that is, after Q queries), uniformly choose a random graph G from R^b_Q. This is the graph constructed by P^b, $b \in 1, 2$.

LEMMA 10. *For every algorithm* ALG*, the process P^1, when interacting with* ALG*, uniformly generates graphs in $\mathcal{G}(n, d)$, and the process P^2, when interacting with* ALG*, uniformly generates graphs in $\mathcal{G}(n/2, n/2, d)$.*

Proof. Consider a specific graph $G \in R^b_0$. Recall that $R^1_0 = \mathcal{G}(n, d)$, and $R^2_0 = \mathcal{G}(n/2, n/2, d)$.

The probability that G is generated by P^b, $b \in 1, 2$, is

$$\Pr[G \in R^b_1] \cdot \Pr[G \in R^b_2 | G \in R^b_1] \cdot \cdots \cdot \Pr[G \in R^b_Q | G \in R^b_{Q-1}] \cdot \frac{1}{|R^b_Q|}$$
$$= \frac{|R^b_1|}{|R^b_0|} \cdot \frac{|R^b_2|}{|R^b_1|} \cdot \cdots \cdot \frac{|R^b_Q|}{|R^b_{Q-1}|} \cdot \frac{1}{|R^b_Q|} = \frac{1}{|R^b_0|},$$

and the lemma follows. ∎

We next want to upper-bound the probability that ALG, after performing $o(n/d)$ queries, gets a positive answer (that is, of the form $(1, *, *)$) when it performs a new vertex-pair query and interacts with either one of the two processes. We first introduce some more notation. For $b \in \{1, 2\}$ and any set of edges $B = \{e_1, \ldots, e_\ell\}$, let $R^b_B \stackrel{\text{def}}{=} R^b_{e_1} \cap \cdots \cap R^b_{e_\ell}$ (where $R^b_{e_j}$ is as defined above in the description of the processes $\{P^b\}_{b \in \{1,2\}}$). Similarly, for any set of edge-pairs $D = \{f_1, \ldots, f_h\}$, let $R^b_{\overline{D}} \stackrel{\text{def}}{=} R_{\overline{f_1}} \cap \cdots \cap R_{\overline{f_h}}$. Finally, let $R^b_{B,\overline{D}} = R^b_B \cap R^b_{\overline{D}}$. That is, $R^b_{B,\overline{D}}$ is the subset of all graphs in R^b that are consistent with a particular set of edges and a particular set of nonedges. In particular, if B and D correspond to the set of edges and nonedges, respectively, that were observed by ALG in the course of its first t queries, then $R^b_{B,\overline{D}} = R^b_t$.

LEMMA 11. *Let $B = \{e_1, \ldots, e_\ell\}$ be any set of edges, and let $D = \{f_1, \ldots, f_h\}$ be any set of edge-pairs such that no edge-pair in D appears in B and such that $|B|, |D| = o(n/d)$. Then for each $b \in \{1, 2\}$ and for every $(u, v, i, j) \notin B$ we have*

$$\frac{|R^b_{(u,v,i,j)} \cap R^b_{B,\overline{D}}|}{|R^b_{B,\overline{D}}|} \leq \frac{4}{dn}.$$

Proof. Consider first the process P^1 and recall that R^1 is the set of graphs in the support of $\mathcal{G}(n, d)$. For the sake of brevity we remove the superscript 1 for the remaining discussion (until we turn to the process P^2). For each $e_k \in B$, let $e_k = (u_k, v_k, i_k, j_k)$. For any $w \leq \ell = |B|$, the total number of possible perfect

matchings in the matching table that match c_{u_k,i_k} to c_{v_k,j_k} for $1 \le k \le w$ (that is, the number of matchings that are consistent with the edges $e_1, \ldots, e_w$) is

$$\frac{(dn-2w)!}{2^{(\frac{dn}{2}-w)}(\frac{dn}{2}-w)!}.$$

The argument required for establishing this expression is a slight extension of the one used in the proof of Lemma 9 (when determining the size of R_0). It follows that for any (u,v,i,j) such that neither $c_{u,i}$ nor $c_{v,j}$ is yet matched,

$$\frac{|R_{(u,v,i,j)} \cap R_B|}{|R_B|} = \frac{\frac{(dn-2(k+1))!}{2^{(\frac{dn}{2}-(k+1))}(\frac{dn}{2}-(k+1))!}}{\frac{(dn-2k)!}{2^{(\frac{dn}{2}-k)}(\frac{dn}{2}-k)!}} = \frac{1}{dn-2k-1}.$$

For an edge-pair (u,v) we can thus deduce that

$$\begin{aligned}\frac{|R_{(u,v)} \cap R_B|}{|R_B|} &= \frac{\left|\left(\bigcup_{1\le i,j\le d} R_{(u,v,i,j)}\right) \cap R_B\right|}{|R_B|} \\ &\le \frac{\sum_{1\le i,j\le d} |R_{(u,v,i,j)} \cap R_B|}{|R_B|} \\ &\le \frac{d^2}{dn-2k-1} = O\left(\frac{d}{n}\right). \end{aligned} \tag{30}$$

Hence we obtain that

$$\begin{aligned}\frac{|R_{B,\overline{D}}|}{|R_B|} &= \frac{\left|R_B \cap \left(\bigcap_{1\le w\le h} R_{\overline{f_w}}\right)\right|}{|R_B|} \\ &= \frac{\left|R_B \setminus \left(\bigcup_{1\le w\le h} R_{f_w}\right)\right|}{|R_B|} \\ &\ge 1 - \sum_{1\le w\le h} \frac{|R_{f_w} \cap R_B|}{|R_B|} \\ &= 1 - o\left(\frac{n}{d}\right)\cdot O\left(\frac{d}{n}\right) = 1 - o(1), \end{aligned} \tag{31}$$

and so $\frac{|R_B|}{|R_{B,\overline{D}}|} = 1 + o(1)$.

Putting the above together, we get that for every $(u,v,i,j) \notin B$

$$\begin{aligned}\frac{|R_{(u,v,i,j)} \cap R_{B,\overline{D}}|}{|R_{B,\overline{D}}|} &= \frac{|R_{(u,v,i,j)} \cap R_B|}{|R_B|} \cdot \frac{|R_{(u,v,i,j)} \cap R_{B,\overline{D}}|}{|R_{(u,v,i,j)} \cap R_B|} \cdot \frac{|R_B|}{|R_{B,\overline{D}}|} \\ &\le \frac{1}{dn-2k-1} \cdot 1 \cdot (1+o(1)) \le \frac{4}{dn}. \end{aligned} \tag{32}$$

We now turn to $P^b = P^2$ and recall that R^2 is the support of $\mathcal{G}(n/2, n/2, d)$. Also recall that in order to construct a random graph from $\mathcal{G}(n/2, n/2, d)$ we do the following: we partition the n vertices into two equal parts at random, and we use two matching tables (one named the odd table and one the even table). We then

randomly select a perfect matching between cells in the odd table and cells in the even table. Here too we remove the superscript $b = 2$ for the sake of brevity.

Let $w = o(n/d)$, and let M_w be the number of possibilities to partition n vertices into two parts of equal size so that the endpoints of the edges $e_1, \ldots, e_w$ belong to different sides of the partition. The total number of graphs in $R = R^2$ that contain the edges $e_1, \ldots, e_w \in B$ is $M_w \cdot (dn/2 - w)!$. To verify this, consider any fixed equal partition amongst the M_w possible ones (such that the endpoints of the edges $e_1, \ldots, e_w$ belong to different sides of the partition). For each such partition, consider the $dn/2 - w$ cells in the odd/even tables which are not part of the edges $e_1, \ldots, e_w$. A matching between them can be uniquely defined by an arbitrary order on the unmatched cells in the odd table and a permutation on the unmatched cells in the even table.

Note that by the definition of M_w we have that $\frac{M_{w+1}}{M_w} \leq 1$. Therefore,

$$\frac{|R_{(u,v,i,j)} \cap R_B|}{|R_B|} = \frac{M_{k+1} \cdot (dn/2 - (k+1))!}{M_k \cdot (dn/2 - k)!} \leq \frac{1}{dn/2 - k}. \tag{33}$$

As in the case of $P^b = P^1$ it follows that

$$\frac{|R_{(u,v)} \cap R_B|}{|R_B|} \leq \frac{d^2}{dn/2 - k} = O\left(\frac{d}{n}\right), \tag{34}$$

from which we can obtain that $\frac{|R_B|}{|R_{B,\overline{D}}|} = 1 + o(1)$.

By combining the above we get that

$$\begin{aligned} \frac{|R_{(u,v,i,j)} \cap R_{B,\overline{D}}|}{|R_{B,\overline{D}}|} &= \frac{|R_{(u,v,i,j)} \cap R_B|}{|R_B|} \cdot \frac{|R_{(u,v,i,j)} \cap R_{B,\overline{D}}|}{|R_{(u,v,i,j)} \cap R_B|} \cdot \frac{|R_B|}{|R_{B,\overline{D}}|} \\ &\leq \frac{1}{dn/2 - k} \cdot 1 \cdot (1 + o(1)) \leq \frac{4}{dn}. \end{aligned} \tag{35}$$

∎

The next two lemmas follow as corollaries of Lemma 11.

LEMMA 12. *Let* ALG *be an algorithm that interacts with a process P^b, $b \in 1, 2$, and performs $o(n/d)$ queries. The probability that* ALG *detects an edge by a vertex-pair query at any step is $o(1)$.*

Proof. Consider the interaction of ALG with P^b. For each $1 \leq t \leq Q$, $Q = o(n/d)$, let B_{t-1} be the set of edges in the induced knowledge graph G_{t-1}, and let D_{t-1} be the set of edge-pairs that are nonedges in the graph. By definition of the process P^b, we know that $R^b_{t-1} = R^b_{B_{t-1},\overline{D_{t-1}}}$. Suppose that the tth query of ALG is a vertex-pair query (u_t, v_t) (where, by our assumption, the pair (u_t, v_t) is neither an edge(-pair) nor a nonedge in G_{t-1}, but other than that it may be any vertex-pair). Then by Lemma 11 (and using the fact that $R^b_{t-1} = R^b_{B_{t-1},\overline{D_{t-1}}}$), for every $i, j \in \{1, \ldots, d\}$, $|R^b_{(u_t,v_t,i,j)} \cap R^b_{t-1}|/|R^b_{t-1}| \leq 4/(dn)$. Therefore,

$$\begin{aligned} \frac{\left|R^b_{(u_t,v_t)} \cap R^b_{t-1}\right|}{|R^b_{t-1}|} &= \frac{\left|\left(\bigcup_{i,j \in \{1,\ldots,d\}} R^b_{u_t,v_t,i,j}\right) \cap R^b_{t-1}\right|}{|R^b_{t-1}|} \\ &\leq \frac{\sum_{i,j=1}^{d} \left|\left(R^b_{u_t,v_t,i,j} \cap R^b_{t-1}\right)\right|}{|R^b_{t-1}|} \leq \frac{4d}{n}. \end{aligned} \tag{36}$$

It follows by the definition of P^b that the answer $a_t = (y_t, i_t, j_t)$ satisfies $y_t = 1$ with probability at most $4d/n$. Note that the above is true for every step t, for every pair

of sets B_{t-1} and D_{t-1}, and for every vertex-pair (u_t, v_t) (that does not already appear in B_{t-1} or in D_{t-1}). Hence, the probability that $y_t = 1$ in any one of the $Q = o(n/d)$ queries performed by the algorithm is $o(1)$ as required. ∎

Next we turn to neighbor queries and prove a similar claim.

LEMMA 13. *Let* ALG *be an algorithm that interacts with process P^b, $b \in 1, 2$, and performs $o(\sqrt{n})$ queries. Then the probability that* ALG*, while performing a neighbor query $q_t = (u_t, i_t)$ at any step t of the interaction, receives as an answer $a_t = (v_t, j_t)$, where v_t is a vertex that belongs to the knowledge graph G_{t-1}, is $o(1)$.*

Proof. Let $t = o(\sqrt{n})$, and let $q_t = (u_t, i_t)$ be a neighbor query. Denote by p_t the probability that following this query ALG reaches a vertex that was previously visited (i.e., p_t is the probability that in the answer $a_t = (v_t, j_t)$ we get a vertex v_t that already belongs to the knowledge graph G_{t-1}). Then by the definition of P^b (for both $b = 1$ and $b = 2$),

$$p_t = \frac{\left|\bigcup_{v \in G_{t-1},\, 1 \le j \le d} \left(R^b_{(u_t, v, i_t, j)} \cap R^b_{t-1}\right)\right|}{|R^b_{t-1}|}. \tag{37}$$

By Lemma 11 (using the same argument as the one applied in the proof of Lemma 12), we can deduce that for any specific v and j, $v \in G_{t-1}$ and $1 \le j \le d$,

$$\left|R^b_{(u_t, v, i_t, j)} \cap R^b_{t-1}\right| \;\le\; \frac{4}{dn} \cdot \left|R^b_{t-1}\right|. \tag{38}$$

Therefore,

$$\begin{aligned}\left|\bigcup_{v \in G_{t-1},\, 1 \le j \le d} \left(R^b_{(u_t, v, i_t, j)} \cap R^b_{t-1}\right)\right| &\le \sum_{v \in G_{t-1},\, 1 \le j \le d} \left|R^b_{(u_t, v, i_t, j)} \cap R^b_{t-1}\right| \\ &\le d \cdot 2t \cdot \frac{4}{dn} \cdot \left|R^b_{t-1}\right| \quad = \quad \frac{8t}{n} \cdot \left|R^b_{t-1}\right|. \end{aligned} \tag{39}$$

Since $t = o(\sqrt{n})$, we get that $p_t = o(1/\sqrt{n})$. Since this holds for every $1 \le t \le Q = o(\sqrt{n})$, the probability that for some t (in which a neighbor query is performed) the algorithm reaches a vertex in the knowledge graph is $o(1)$. ∎

In particular, we can obtain the following corollary.

COROLLARY 9. *Let* ALG *be an algorithm that interacts with process P^b and performs $Q = o(\min(\sqrt{n}, n/d))$ queries. Then* ALG *does not detect a cycle with probability $1 - o(1)$.*

Recall that D^b_{ALG}, $b \in \{1, 2\}$, denotes the distribution on query-answer histories (of length Q), induced by the interaction of ALG and P^b. We are now ready to show that the two distributions are indistinguishable if Q is sufficiently small.

LEMMA 14. *For every algorithm* ALG *that asks $Q = o(\min(\sqrt{n}, n/d))$ queries, the statistical distance between D^1_{ALG} and D^2_{ALG} is at most $o(1)$. Furthermore, with probability at least $1 - o(1)$, the knowledge graph at the time of termination of* ALG *contains no cycles.*

Proof. Recall that G_Q, the knowledge graph after Q queries, contains all the vertices that appeared in one of the queries or answers, all the edges found by queries, and all the nonedges found by queries.

Recall that we assume without loss of generality that ALG does not ask queries whose answer can be derived from the knowledge graph, since those give it no new information. Under this assumption the following hold:

1. According to Lemma 12, both in D^1_{ALG} and in D^2_{ALG} the total weight of query-answer histories in which an edge is detected by a vertex-pair query is $o(1)$. In all other histories (whose weight is $1-o(1)$ under both distributions), all answers to vertex-pair queries are of the form $(0,1,1)$.
2. According to Lemma 13, both in D^1_{ALG} and in D^2_{ALG} the total weight of query-answer histories in which any neighbor query q_t is answered with a vertex that belongs to the knowledge graph G_{t-1} is $o(1)$. In particular, this means that, with probability at least $1-o(1)$, the knowledge graph at the time of termination of ALG contains no cycles.

Let Π_1 be the set of all possible query-answer histories in which a vertex-pair query is answered by $(1,*,*)$, let Π_2 be the set of histories in which a neighbor query is answered with a vertex that appears in the current knowledge graph, and let Π_3 be the set of all remaining histories. Then the statistical distance between D^1_{ALG} and D^2_{ALG} is upper-bounded by

$$\sum_{\pi\in\Pi_1}\left|D^1_{\mathrm{ALG}}(\pi)-D^2_{\mathrm{ALG}}(\pi)\right|+\sum_{\pi\in\Pi_2}\left|D^1_{\mathrm{ALG}}(\pi)-D^2_{\mathrm{ALG}}(\pi)\right| + \sum_{\pi\in\Pi_2}\left|D^1_{\mathrm{ALG}}(\pi)-D^2_{\mathrm{ALG}}(\pi)\right|. \tag{40}$$

Observe that

$$\begin{aligned}\sum_{\pi\in\Pi_1}\left|D^1_{\mathrm{ALG}}(\pi)-D^2_{\mathrm{ALG}}(\pi)\right| &\le \sum_{\pi\in\Pi_1}D^1_{\mathrm{ALG}}(\pi)+\sum_{\pi\in\Pi_1}D^2_{\mathrm{ALG}}(\pi)\\ &= D^1_{\mathrm{ALG}}(\Pi_1)+D^2_{\mathrm{ALG}}(\Pi_2)\end{aligned}$$

(and similarly for the sum over Π_2). Hence the first two terms in (40) contribute $o(1)$.

As for the third term, first note that for every fixed history $\langle(q_1,a_1),\ldots,(q_{t-1},a_{t-1})\rangle$, the distribution on the next query q_t that ALG performs is the same no matter which process it interacts with. By definition of Π_3, if q_t is a vertex-pair query, then $a_t=(0,1,1)$ for both processes. Finally, if $q_t=(u_t,i_t)$, then again by the definition of Π_3 the answer $a_t=(v_t,j_t)$ is such that $v_t\notin G_{t-1}$. For both processes, conditioned on $v_t\notin G_{t-1}$, the vertex v_t is uniformly distributed among all vertices not in G_{t-1}, and j_t is uniformly distributed over $\{1,\ldots,d\}$. Therefore, the third term in (40) is bounded by $|D^1_{\mathrm{ALG}}(\Pi_3)-D^2_{\mathrm{ALG}}(\Pi_3)|$ which is $o(1)$ as well, and the lemma follows. □

Theorem 8 follows by combining Lemma 14 with Lemma 9.

5.1. Self-loops and multiple edges. The lower-bound proof as stated above is valid for graphs that may contain multiple edges and self-loops. In both the distribution $\mathcal{G}(n,d)$ and the distribution $\mathcal{G}(n/2,n/2,d)$ the probability of a multiple edge between vertices u and v is $O(\frac{d^2}{n^2})$, and the probability of a self-loop edge incident to vertex v is $O(\frac{d}{n})$. Thus, graphs created according to these distributions contain self-loops and multiple edges with probability close to 1. However, with probability $1-o(1)$ there are at most $O(d^2)$ loops and multiple edges in the graphs created according to these distributions.

We have shown in Lemma 9 that graphs created according to the distribution $\mathcal{G}(n,d)$ are ϵ-far from being bipartite with probability $1-o(1)$. Hence we can deduce that by removing self-loops and multiple edges from a graph G constructed according to a distribution $\mathcal{G}(n,d)$, the resulting graph is ϵ-far from being bipartite with probability $1-o(1)$.

In addition, an algorithm ALG that interacts with process P^b detects a multiple edge with probability $o(1)$ due to the following reason: ALG does not detect any edges by sampling steps with probability $1 - o(1)$. The probability to detect a multiple edge/self-loop in a neighbor query is at most the probability that such a query is answered by a vertex in the knowledge graph. This probability was shown (in Lemma 13) to be $o(1)$.

Thus, P^b in the end of the interaction with ALG can delete all the loops and multiple edges from the resulting graph so that the resulting graph contains no loops and multiple edges. In the case of P^2 the graph is bipartite as before, and in the case of P^1 the graph is (still) ϵ-far from being bipartite.

As a conclusion we get that our lower bound is valid also for graphs with no multiple edges and loops.

5.2. The necessity of both neighbor queries and vertex-pair queries. As noted in the introduction, our lower-bound proof implies the necessity of both neighbor queries and vertex-pair queries in obtaining an upper bound whose dependence on n and m is $\tilde{O}(\min(\sqrt{n}, n^2/m))$. Specifically, if only neighbor queries are allowed, then Lemma 13 implies a lower bound of $\Omega(\sqrt{n})$ for every m, which is higher than $\tilde{O}(n^2/m)$ when $m = \omega(n^{1.5})$. On the other hand, every bipartite tester that uses only vertex-pair queries has to make $\Omega(n^2/m)$ since otherwise it will not see any edge. This lower bound is above the upper bound of $\tilde{O}(\sqrt{n})$ when $m = o(n^{1.5})$.

5.3. A lower bound for testing *k*-colorability. It is possible to generalize the lower bound stated for bipartite testers to a lower bound for k-colorability. By using similar arguments we can get that a graph chosen uniformly from the distribution $\mathcal{G}(n, d)$ is ϵ-far from being k-colorable. In addition, by defining a distribution $\mathcal{G}(n/k, n/k, \ldots, n/k, d)$ in an analogous way to the definition of $\mathcal{G}(n/2, n/2, d)$ and by an analogous definition of the processes P^b, $b \in 1, 2$, we can get a lower bound for k-colorability which is $\Omega(\min(\sqrt{n}, n^2/m))$.

Appendix. A formal definition of $\mathbf{M}_{\ell_1}^{\ell_2}(H)$. For every vertex v in H we have a state v in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ (see Figure 8). For simplicity, we shall continue referring to these states as vertices. Let the *border* of H, denoted $B(H)$, be the set of vertices in H that have at least one neighbor in G that is not in H. Then, for every vertex $v \in B(H)$, we have a set $a_{v,1}, \ldots, a_{v,\ell_1}$ of *auxiliary* states. Let $p_{v,u}^H(t)$ denote the probability of a walk of length t that starts at v and ends at u without passing through any other vertex in H. Namely, it is the sum over all such walks w, of the product, taken over all steps in w, of the transition probabilities of these steps. In particular, $p_{v,v}^H(1) \geq \frac{1}{2}$ (where equality holds in case v has degree d), and for every $u \in \Gamma(v)$, $p_{v,u}^H(1) = \frac{1}{2d}$ (here we assume that we can choose a random neighbor of a vertex within time which is $O(1)$). The transition probabilities, $q_{x,y}$, in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ are defined as follows:

- For every v and u in H, $q_{v,u} = \sum_{t=1}^{\ell_2 - 1} p_{v,u}^H(t)$.
  Thus, $q_{v,u}$ is a sum of $p_{v,u}^H(1)$ and $\sum_{t=2}^{\ell_2-1} p_{v,u}^H(t)$. The first term implies that for every v in H, $q_{v,v} \geq \frac{1}{2}$ and for every pair of neighbors v and u, $q_{v,u} \geq \frac{1}{2d}$. The second term, which we refer to as the *excess* probability, is due to walks of length less than ℓ_2 (from v to u) passing through vertices outside of H and can be viewed as a *contraction* of these walks.
  Hence, for every pair of vertices v and u, $q_{v,u} = q_{u,v}$.
- For every $v \in B(H)$, $q_{v,(a_{v,1})} = \sum_{u \in H} \sum_{t \geq \ell_2} p_{v,u}^H(t)$; for every ℓ, $1 \leq \ell < \ell_1$, $q_{(a_{v,\ell}),(a_{v,\ell+1})} = 1$; and for every $u \in H$, $q_{(a_{v,\ell_1}),u} = \frac{1}{q_{v,(a_{v,1})}} \cdot \sum_{t \geq \ell_2} p_{v,u}^H(t)$.

(The parentheses added in the notation above (e.g., $q_{(a_{v,\ell}),(a_{v,\ell+1})}$) are only for the sake of readability.)

In other words, $q_{v,(a_{v,1})}$ is the probability that a random walk in G that starts from v takes at least ℓ_2 steps outside of H before returning to H, and $q_{(a_{v,\ell_1}),u}$ is the conditional probability of reaching u in such a walk. Thus, the auxiliary states form auxiliary paths in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$, where these paths correspond to walks of length at least ℓ_2 outside of H.

We shall restrict our attention to walks of length at most ℓ_1 in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$, and hence any walk that starts at a vertex of H and enters an auxiliary path never returns to vertices of H.

For any two states y, z in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ let $q_{y,z}(t)$ be the probability that a walk of length t starting from y ends at z. In particular, $q_{y,z} \equiv q_{y,z}(1)$, and for any two vertices u and v and any integer t, we have $q_{u,v}(t) = q_{v,u}(t)$. We further let the parity of the lengths of paths corresponding to walks in G be carried on to $\mathrm{M}_{\ell_1}^{\ell_2}(H)$. That is, each transition between vertices v and u that corresponds to walks outside of H consists of two transitions—one due to even-length paths corresponding to walks from v to u outside of H and one due to odd-length paths. For any two vertices in H we let $q_{v,u}^b(t)$ denote the probability in $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ of a walk of length t starting from v, ending at u, and corresponding to a path whose length has parity b.

In all that follows we assume that G is connected. Our analysis can easily be modified to deal with the case in which G is not connected, simply by treating separately each of its connected components. Under the assumption that G is connected, for every v and u in H, there exists a t such that $q_{u,v}(t) > 0$, and hence $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ is irreducible. Furthermore, because for each $v \in H$ it holds that $q_{v,v} \geq \frac{1}{2}$, $\mathrm{M}_{\ell_1}^{\ell_2}(H)$ is also aperiodic. Thus it has a unique stationary distribution.

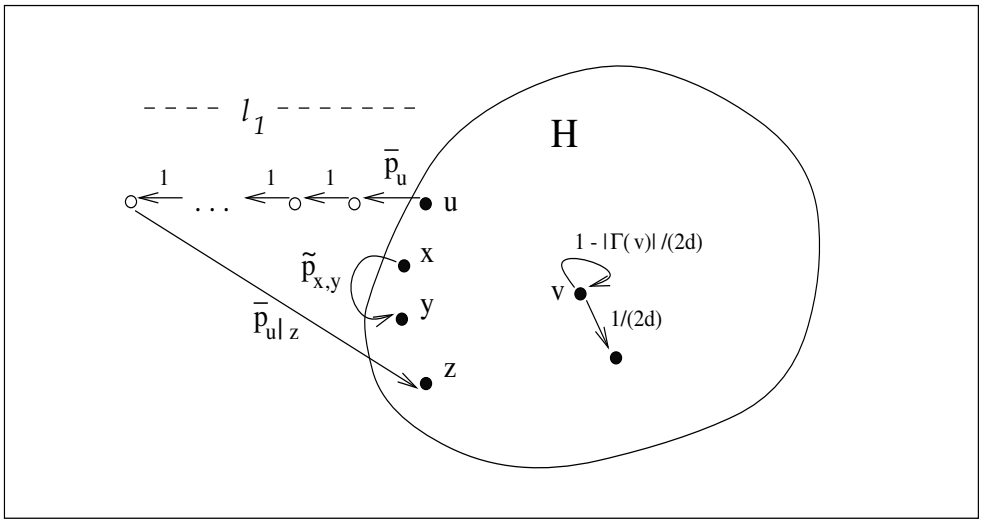


FIG. 8. *The structure of* $\mathrm{M}_{\ell_1}^{\ell_2}(H)$. *The states corresponding to vertices of* H *are depicted as black dots, and the auxiliary states are depicted as white ones. Here* $\tilde{p}_{x,y}$ *denotes the transition probability between any two vertices* $x, y \in B(H)$, *which equals* $\sum_{t=1}^{\ell_2-1} p_{x,y}^H(1)$, $\bar{p}_u$ *denotes the probability of entering an auxiliary path starting from* $u \in B(H)$, *which equals* $\sum_{z\in H}\sum_{t\geq\ell_2} p_{u,z}^H(t)$, *and* $\bar{p}_{u|z}$ *denotes the probability of returning from the last state on this auxiliary path to* $z \in B(H)$, *which equals* $\frac{1}{\bar{p}_u} \cdot \sum_{t\geq\ell_2} p_{u,z}^H(t)$.

Acknowledgment. We would like to thank the anonymous reviewers of this paper for very helpful comments.

REFERENCES

[1] N. Alon, *Testing subgraphs of large graphs*, Random Structures Algorithms, 21 (2002), pp. 359–370.
[2] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, *Efficient testing of large graphs*, Combinatorica, 20 (2000), pp. 451–476.
[3] N. Alon and M. Krivelevich, *Testing k-colorability*, SIAM J. Discrete Math., 15 (2002), pp. 211–227.
[4] N. Alon and A. Shapira, *Testing subgraph in directed graphs*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 700–709.
[5] N. Alon and J. H. Spencer, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.
[6] M. Bender and D. Ron, *Testing properties of directed graphs: Acyclicity and connectivity*, Random Structures Algorithms, 20 (2002), pp. 184–205.
[7] A. Bogdanov, K. Obata, and L. Trevisan, *A lower bound for testing 3-colorability in bounded-degree graphs*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 93–102.
[8] A. Bogdanov and L. Trevisan, *Lower Bounds for Testing Bipartiteness in Dense Graphs*, Tech. report TR02-064, Electronic Colloquium on Computational Complexity (ECCC), 2002. Available from http://www.eccc.uni-trier.de/eccc/.
[9] B. Bollobas, P. Erdos, M. Simonovitz, and E. Szemeredi, *Extremal graphs without large forbidden subgraphs*, Ann. Discrete Math., 3 (1978), pp. 29–41.
[10] A. Czumaj and C. Sohler, *Abstract combinatorial programs and efficient property testers*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 83–92.
[11] U. Feige, *On sums of independent random variables with unbounded variance, and estimating the average degree in a graph*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 594–603.
[12] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.
[13] O. Goldreich and D. Ron, *A sublinear bipartite tester for bounded degree graphs*, Combinatorica, 19 (1999), pp. 335–373.
[14] O. Goldreich and D. Ron, *Property testing in bounded degree graphs*, Algorithmica, 32 (2002), pp. 302–343.
[15] O. Goldreich and D. Ron, *On Estimating the Average Degree of a Graph*, Tech. report TR04-013, Electronic Colloquium on Computational Complexity (ECCC), 2004. Available from http://www.eccc.uni-trier.de/eccc/.
[16] O. Goldreich and L. Trevisan, *Three theorems regarding testing graph properties*, Random Structures Algorithms, 23 (2003), pp. 23–57.
[17] A. Lubotzky, R. Phillips, and P. Sarnak, *Explicit expanders and the Ramanujan conjectures*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, ACM, New York, 1986, pp. 240–246.
[18] G. A. Margulis, *Explicit constructions of expanders*, Problemy Peredachi Informatsii, 9 (1973), pp. 71–80 (in Russian).
[19] M. Mihail, *Conductance and convergence of Markov chains: A combinatorial treatment of expanders*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 526–531.
[20] M. Parnas and D. Ron, *Testing the diameter of graphs*, Random Structures Algorithms, 20 (2002), pp. 165–183.
[21] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.
[22] A. Yao, *Probabilistic computation: Towards a unified measure of complexity*, in Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 33, No. 6, pp. 1484–1512

DEPTH OPTIMAL SORTING NETWORKS RESISTANT TO k PASSIVE FAULTS*

MAREK PIOTRÓW†

Abstract. We study the problem of constructing a sorting network that is tolerant to faults and whose running time (i.e., depth) is as small as possible. We consider the scenario of worst-case comparator faults and follow the model of *passive* comparator failure proposed by Yao and Yao [*SIAM J. Comput.*, 14 (1985), pp. 120–128], in which a faulty comparator outputs its inputs directly without comparison. Our main result is the first construction of an N-input k-fault-tolerant sorting network with an asymptotically optimal depth $\theta(\log N+k)$. That improves over the result of Leighton and Ma [*Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, Velen, Germany, 1993, ACM, New York, pp. 30–41], whose network is of depth $O(\log N+k\log\frac{\log N}{\log k})$.

Actually, we present a fault-tolerant correction network that can be added after any N-input sorting network to correct its output in the presence of at most k faulty comparators. Since the depth of the network is $O(\log N + k)$ and the constants hidden behind the "O" notation are small, the construction can be of practical use.

Developing the techniques necessary to show the main result, we construct a fault-tolerant network for the insertion problem. As a by-product, we get an N-input $O(\log N)$-depth INSERT-network that is tolerant to random faults, thereby answering a question posed by Ma in his Ph.D. thesis [*Fault-Tolerant Sorting Network*, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, 1994].

The results are based on a new notion of constant delay comparator networks, that is, networks in which each register is used (compared) only in a period of time of a constant length. Copies of such networks can be pipelined with only a constant increase in the total depth per copy.

Key words. fault-tolerant sorting, sorting networks, comparators

AMS subject classifications. 68Q05, 68Q25

DOI. 10.1137/S0097539799354308

1. Introduction.

A comparator is a simple device that takes the contents of two registers and performs a *compare-exchange* operation on them; that is, the minimum is put into the first register and the maximum into the second one. Networks built from comparators are commonly used to perform such tasks as selection, sorting, and merging. They have also proved to be very useful for a variety of applications, including circuit switching and packet routing [6]. Therefore, there has been much research in this area. The most famous constructions are those of Batcher [3], who introduced two types of sorting networks both of depth $(\log^2 N + \log N)/2$, and of Ajtai, Komlos, and Szemeredi [1], who showed that there exist sorting networks of depth $O(\log N)$.[1] None of these constructions addresses the issue of fault-tolerance.

The study of fault-tolerant sorting networks was initiated by Yao and Yao [13] in 1985. They introduced a fault type in which a *faulty* comparator does not work at all: the contents of registers remain unchanged. Later, this type of fault was called

*Received by the editors April 2, 1999; accepted for publication (in revised form) June 4, 2004; published electronically September 2, 2004. This research was partially supported by the Alexander von Humboldt Stiftung. A preliminary version of this paper appeared in *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms* (*SODA*'96), Atlanta, GA, 1996, SIAM, Philadelphia, PA, pp. 242–251. Preparation of the full version of this paper was supported by grant KBN 8T 11C 029 13.

http://www.siam.org/journals/sicomp/33-6/35430.html

†Institute of Computer Science, University of Wrocław, Przesmyckiego 20, PL-51-151 Wrocław, Poland (mpi@ii.uni.wroc.pl).

[1] In this paper all the logarithms are taken in base 2.

passive. In [13], the authors consider the use of redundancy to enhance reliability for sorting and related networks under two alternative fault models. In the *random-fault* model, comparators fail independently with probability upper-bounded by a constant less than 1, and the goal is to construct networks which work correctly with probability at least $1 - \frac{1}{N}$ under this assumption. In the *k-fault* model (or *worst-case* model), the goal is to build k-fault-tolerant networks, that is, networks which work correctly if any set of at most k comparators is faulty.

Yao and Yao observed that any sorting network could be made random-fault-tolerant if each of the original comparators was replicated $O(\log N)$ times. This yields a fault-tolerant sorting network with $O(\log^2 N)$ depth. No improvement of this bound was obtained for several years. In 1993, Leighton and Ma (see [7]) gave a new analysis of the AKS network [1], which resulted in an improved upper bound $O(\log N \log\log N)$ on the depth of random-fault-tolerant sorting networks. Later, Ma [10] presented a new construction of such networks with optimal size $O(N \log N)$. This result resolved the old conjecture of Yao and Yao that the size of random-fault-tolerant sorting networks is $\omega(N \log N)$.

Other types of faulty comparators were also studied in [2, 7, 8] in this context. In the *reversal* fault model, a faulty comparator outputs its inputs in reversed order. In the *destructive* fault model, a faulty comparator with inputs x and y may output any of the pairs (x, y), (y, x), (x, x), or (y, y); thus it can destroy one of the inputs, replacing it with the other one. In order to tolerate such types of faulty elements, Assaf and Upfal [2] extended the model of comparator networks by allowing the use of more registers than input items and introducing *replicators* that are used to copy the contents of one register to another register. Replicators are assumed to be fault-free. In [2] they gave a general method for converting any sorting network into a reversal-fault-tolerant or destructive-fault-tolerant sorting network in the random-fault model. Applying this technique to the AKS network, they obtained fault-tolerant networks of size $O(N \log^2 N)$. For the destructive fault model the result is asymptotically optimal—the corresponding lower bound was proved by Leighton and Ma in [8]. For the reversal fault model, the upper bound was improved in [7] to $O(N \log^{\log_2 3} N)$. All the mentioned results concern the random fault model. For the worst-case model, a construction of a k-reversal-fault-tolerant sorting network of size $O(N(\log N + k \log\log_k N + k^{\log_2 3}))$ was also provided in [7].

For the worst-case passive model, Yao and Yao [13] constructed a k-fault-tolerant sorting network of the optimal size. They did not consider the depth of these networks; in fact, the increase in the depth was $O(kN)$. Schimmler and Starke [12] gave another construction of the network, where the depth of an N-input k-fault-tolerant sorting network was reduced to $O(k \log N)$. Further improvement was obtained by Leighton, Ma, and Plaxton [7], who derived a new upper bound $O(\log N + k \log \frac{\log N}{\log k})$ on the depth of the networks in question. In this construction, the constant in the "O" notation depends on the structure of a modified AKS-network and is quite big. Our main result closes the gap between this upper bound and the trivial lower bound $\Omega(\log N + k)$.

Actually, we present a fault-tolerant correction network that can be added after any N-input sorting network to correct its output in the presence of at most k faulty comparators. Since the depth of this additional network is $O(\log N + k)$ and the constants hidden behind the "O" notation are not big, the construction can be of practical use.

MAIN THEOREM. *There exists an explicit construction of N-input k-fault-tolerant sorting networks of depth $O(\log N + k)$.*

Developing the techniques necessary to show the main result, we construct a fault-tolerant network for the insertion problem. As a by-product, we get an N-input $O(\log N)$-depth INSERT-network that is tolerant to random faults, thereby answering a question posed by Ma in his Ph.D. thesis [9].

THEOREM 1.1. *There exists an explicit construction of an N-input k-fault-tolerant* INSERT*-network of depth $O(\log N + k)$.*

THEOREM 1.2. *There exists an explicit construction of an N-input random-fault-tolerant* INSERT*-network of depth $O(\log N)$.*

The results are based on a new notion of constant delay comparator networks, that is, networks in which each register is used (compared) only in a period of time of a constant length. Copies of such networks can be pipelined with only a constant increase in the total depth per copy. To look at this method more closely, consider the case of two copies A_1 and A_2 of a d-delay network A. Obviously, A_1 and A_2 have the same pattern of comparators. Therefore, to start performing comparisons from A_2 we do not have to wait until all comparisons from A_1 take place. Indeed, an initial comparator $[i : j]$ from A_2 can be applied just after all comparisons from A_1 involving registers i and j have been finished. Since a copy of $[i : j]$ is also the first comparator applied to i and j in A_1 and since registers i and j are used only in the next $d-1$ phases of A_1, $[i : j]$ can be applied again, this time as a comparator of A_2, just after d phases. The situation is quite similar to the pipelining of data items in systolic computations.

1.1. Overview of the construction. First, let us recall two well-known facts. The first one is the zero-one principle: a network sorts all sequences when it sorts all sequences built from zeroes and ones. The second one is due to Yao and Yao [13]: when any i comparators from a sorting network are faulty and we input a 0-1 sequence to the faulty network, then the Hamming distance[2] between the output and the sorted sequence is at most $2i$. Assuming that the input contains exactly j zeroes and $N-j$ ones, the second fact implies that in the output there are at most i ones inside the initial j items (we will denote them by 1^*s) and at most i zeroes inside the last $N-j$ items (we will denote them by 0^*s). We can look at this sequence as if it were created from a sorted one, $0^j 1^{N-j}$, in two phases: (1) exactly i zeroes were replaced by 1^*s, (2) exactly i ones from the last $N-i$ items were replaced by 0^*s. We would like to correct results of these phases separately. Therefore, the basic structure of our network can be presented as follows, where SORT denotes any sorting network:

Faulty comparators can appear in any part of the network, but their total number must be at most k. If i of them, $0 < i \le k$, appear in the SORT part, at most $k-i$ can appear in the CORRECT 1^*s or CORRECT 0^*s parts. Thus, an A is defined to be a *$CORRECT_k^1$ network* (*$CORRECT_k^0$ network*, respectively) if for all i, $0 < i \le k$, A is $(k-i)$-fault-tolerant and sorts any 0-1 sequence that can be obtained from a sorted one by changing exactly i zeroes to ones (i ones to zeroes, respectively). Our first observation is that parts two and three of the network are symmetric: each one can be obtained from the other by complementing every comparator (recall from [12] that the complement of a comparator $[u : v]$ is defined to be $\overline{[u : v]} = [N-1-v : N-1-u]$).

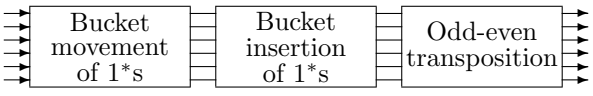
[2]The Hamming distance between two binary sequences of equal length is the number of positions at which the sequences differ.

The second important observation is that after applying both parts, the output sequence is sorted. The reasons are the following: the input to the third part starts with at least $j - i$ zeroes and contains at most i 0^*s among ones; symmetrically, we can analyze the second part as if its input ended up with $N - j$ ones and contained at most i 1^*s among zeroes. We can do that while, whenever 1^* is compared/exchanged with 0^*, the Hamming distance decreases immediately by 2. Both observations are formally proved in section 3. Thus, the only difficult part that remains is a construction of $\mathrm{CORRECT}_k^1$ networks.

We can look at the correction problem as at the problem of insertion: a certain number of "displaced" items (1^*s) should be inserted into a sorted sequence. Since the initial positions of 1^*s are unknown, we split the insertion task into two subtasks: moving 1^*s to certain "well-known" positions and inserting the items from such positions.

In the construction of Schimmler and Starke [12], we can split the whole k-correction network into a sequence of k pairs of subnetworks, where the first element of each pair does the moving subtask and the second element does the insertion. The goal of the whole sequence is to correct an output of a sorting network with i, $i \leq k$, faults and to be resistant to $k - i$ faulty comparators itself. Since the number of fault-free pairs of subnetworks is not smaller than the number of displaced items, a fault-free pair can be assigned to each such item, and the items are corrected one after another, starting from the innermost in an input. The only problem that remains is that the pairs of subnetworks cannot be pipelined well—the total depth of the network is k times the depth of each pair, which yields $O(k \log N)$. We would like to solve the problem by pipelining separately k copies of a modified "moving" subnetwork first, and then k copies of a modified "inserting" subnetwork.

As far as we know, even in the case of one "displaced" item, no fault-tolerant insertion network of the optimal depth has been given so far in the literature. We present such a network in section 4. Then we modify the construction to deal with a greater number of such items. The problem that must be solved is how to avoid, in the process of insertion, comparisons between 1^*s. The result of each such comparison is the same as if the upper 1^* met a faulty comparator, and there can be up to $k/2$ such comparisons in each stage of insertion. The solution that we use is to group each k consecutive registers into a so-called bucket, in each bucket find the minimum and the maximum, and apply the insertion process on buckets, where a comparison between buckets u and v means a comparison between the maximum of u and the minimum of v. Due to the bucket size, the compared items cannot both be 1^* and, therefore, comparisons between 1^*s can happen only inside buckets. The last thing we do after the bucket insertion is to sort items inside buckets (in fact, in one bucket) using odd-even transposition. Thus the whole structure of $\mathrm{CORRECT}_k^1$'s networks can be drawn as follows:



The first two parts are of depth $O(\log N + k)$; the last one is of depth $3k$. The details of their construction and the analysis of their correctness are given in the following sections. It should be emphasized that, in order to make our presentation simpler, we do not force the constants to be optimally small.

2. Preliminaries. Let $N \geq 2$ be an integer and $\mathcal{R}^N$ be the set of N-element vectors of reals. For every $\overline{x} \in \mathcal{R}^N$, $\overline{x}[i]$ denotes the ith term of $\overline{x}$. Let $[N]$ denote the set $\{0, 1, \ldots, N-1\}$, and for $i, j \in [N]$ the *comparator* $[i : j]$ is a mapping from $\mathcal{R}^N$ to $\mathcal{R}^N$ which transforms a vector $\overline{x}$ into vector $\overline{x}' = \overline{x}[i : j]$ defined as follows:

$$\overline{x}'[k] = \begin{cases} \overline{x}[k], & k \neq i, j, \\ \min(\overline{x}[i], \overline{x}[j]), & k = i, \\ \max(\overline{x}[i], \overline{x}[j]), & k = j. \end{cases}$$

Thus $[i : j]$ compares $\overline{x}[i]$ with $\overline{x}[j]$ and places the smaller of them in position i and the larger one in position j. We call $[i : j]$ a *standard comparator* if $i < j$. For a standard comparator $[i : j]$, let $j - i$ be its *length*, and we will sometimes say that it is an *outgoing* comparator from the register i and an *incoming* comparator to the register j.

An *N-comparator stage* S is a set of r comparators $\{[i_1 : j_1], [i_2 : j_2], \ldots, [i_r : j_r]\}$, where $i_1, j_1, \ldots, i_r, j_r \in [N]$ are pairwise different. The comparator stage S defines a *mapping* from $\mathcal{R}^N$ to $\mathcal{R}^N$, which is a composition of the mappings associated with $[i_1 : j_1], [i_2 : j_2], \ldots, [i_r : j_r]$ applied in arbitrary order. Let $\mathit{support}(S)$ denote the set $\{i_1, j_1, i_2, j_2, \ldots, i_r, j_r\}$.

An N-input *comparator network* A is a sequence $S_1, S_2, \ldots, S_d$ of N-comparator stages. The N-input network A defines the mapping from $\mathcal{R}^N$ to $\mathcal{R}^N$ by successively applying the mappings induced by $S_1, S_2, \ldots, S_d$. Let $\overline{x}A$ denote the result of applying A to $\overline{x} \in \mathcal{R}^N$. A *standard network* is a network consisting of standard comparators only. Let $\mathit{depth}(A) = d$ and $\mathit{size}(A) = \sum_{i=1}^{d} |S_i|$.

For an N-input network $A = S_1, S_2, \ldots, S_d$ and $j \in [N]$, we define the following parameters:

$$\begin{aligned} \mathit{fst}(j, A) &= \min\{1 \leq i \leq d : j \in \mathit{support}(S_i)\}, \\ \mathit{lst}(j, A) &= \max\{1 \leq i \leq d : j \in \mathit{support}(S_i)\}, \\ \mathit{delay}(A) &= \max_{j \in [N]} \{\mathit{lst}(j, A) - \mathit{fst}(j, A) + 1\}. \end{aligned}$$

We can look at a computation in a comparator network A as the process of synchronous data movement through an array of computation units (i.e., comparators), where each stage is a column of the array, and rows correspond to registers. We assume that it takes a unit of time to move data items to the next column. In this way, we can consider the whole process as systolic computation. In this context, $\mathit{fst}(j, A)$ denotes the first unit of time when the item in register j reaches a computation unit, and $\mathit{delay}(A)$ is the maximum time required to move any data item through all its computation units. At time $\mathit{fst}(j, A) + \mathit{delay}(A)$, the contents of register j is fixed after all computations and can possibly be pipelined to a next copy of A.

All comparator networks presented in this paper are standard, and whenever it is possible in the constructions we use subnetworks of a constant delay. This allows us to put copies of such a subnetwork one after another so that the total depth is small. The following definitions and Proposition 2.1 formalize this technique.

Let $A = S_1, S_2, \ldots, S_d$ and $A' = S'_1, S'_2, \ldots, S'_{d'}$ be N-input comparator networks such that for each i, $1 \leq i \leq \min(d, d')$, $\mathit{support}(S_i) \cap \mathit{support}(S'_i) = \emptyset$. Then $A \cup A'$ is defined to be $(S_1 \cup S'_1), (S_2 \cup S'_2), \ldots, (S_{\max(d,d')} \cup S'_{\max(d,d')})$, where empty stages are added at the end of the network of smaller depth. Let $A^{\Rightarrow D}$ denote the shift of A

by D stages, that is,

$$A^{\Rightarrow D} = \underbrace{\emptyset, \ldots, \emptyset}_{D \text{ times}}, S_1, S_2, \ldots, S_d.$$

PROPOSITION 2.1. *Let $A = S_1, S_2, \ldots, S_d$ be an N-input comparator network. Then the network $B \stackrel{\text{def}}{=} A \cup A^{\Rightarrow delay(A)}$ is correctly defined, and $depth(B) = depth(A) + delay(A)$.*

Proof. Let $D = delay(A)$. To prove that B is correctly defined, it suffices to check whether

$$\forall\ 0 \le j \le N-1, \quad lst(j, A) < fst(j, A^{\Rightarrow D}).$$

Since for each j, $0 \le j \le N-1$, $fst(j, A^{\Rightarrow D}) = D + fst(j, A)$ and $lst(j, A) - fst(j, A) < D$, the statement holds and, therefore, B is correctly defined. The second part of the lemma follows directly from the definition. □

Using Proposition 2.1, we can define a network $A^{(i)}$ built from i, $i > 0$, copies of the network A, where copies are put one after another, each shifted further by $delay(A)$ stages. Formally,

$$A^{(i)} \stackrel{\text{def}}{=} \bigcup_{j=0}^{i-1} A^{\Rightarrow j \cdot delay(A)}.$$

Most of our networks will be defined recursively. To make recursive definitions precise, the standard set of register indexes $\{0, 1, \ldots, N-1\}$ will sometimes be replaced by another N-element set of nonnegative integers, and comparators will be redefined to this set. We will also use a one-to-one order-preserving mapping to replace one set of indexes by another. For example, if A is a comparator network and s is a positive integer, then $A \Downarrow s$ will denote the network whose registers are indexed by $\{s, s+1, \ldots, s+N-1\}$, and each comparator $[i : j]$ in A is replaced by $[i+s : j+s]$.

An N-input network A is an *INSERT-network* if, for every $\overline{x} \in \mathcal{R}^N$, the fact that $(\overline{x}[1], \ldots, \overline{x}[N-1])$ is sorted implies that $\overline{y} = \overline{x}A$ is sorted. An N-input network A is a *MAX-network* (*MINMAX-network*) if, for every $\overline{x} \in \mathcal{R}^N$, we have $(\overline{x}A)[N-1] = \max\{\overline{x}[0], \ldots, \overline{x}[N-1]\}$ (respectively, if A is a MAX-network and $(\overline{x}A)[0] = \min\{\overline{x}[0], \ldots, \overline{x}[N-1]\}$ for every $\overline{x} \in \mathcal{R}^N$ holds).

We say that B is a j-fault subnetwork of A if B can be obtained from A by deleting exactly j comparators. By definition of a faulty comparator, instead of deleting comparators it is equivalent to say that the comparators in question fail. An N-input SORT-network (MAX-network, MINMAX-network, INSERT-network) is said to be k-fault-tolerant if every j-fault subnetwork of A, $j \le k$, is also a SORT-network (MAX-network, MINMAX-network, INSERT-network, respectively).

The following tool is very useful in the studies of comparator networks.

PROPOSITION 2.2 (zero-one principle). *If an N-input network sorts all 2^N sequences of 0's and 1's, then it is a SORT-network.*

Proof. See [4, 5.3.4, Theorem Z]. □

The zero-one principle can easily be generalized to the case of INSERT-networks, MAX-networks, and MINMAX-networks. Using the zero-one principle, we can restrict our attention to input vectors $\overline{x} \in \{0, 1\}^N$.

PROPOSITION 2.3 (insert 0-1 principle). *If a standard N-input network sorts all N sequences of the form 10^t1^{N-t-1}, $0 \le t < N$, then it is an INSERT-network.*

Proof. The proof follows the same ideas as the proof of the zero-one principle. Recall that for any nondecreasing function f, a comparator network A, and an input $\overline{x}$, $f(\overline{x})A = f(\overline{x}A)$. Let A be a standard network that sorts all N sequences of the form 10^t1^{N-t-1}, $0 \le t < N$. To obtain a contradiction, suppose that there is an input sequence $\overline{x} \subset \mathcal{R}^N$ such that $(x[1], x[2], \ldots, x[N-1])$ is sorted but $\overline{y} = \overline{x}A$ is not sorted. Let i be the index such that $y[i] > y[i+1]$, and define $f(z)$ to be 1 if $z \ge y[i]$ and 0 otherwise. Then $f(\overline{y})$ is not sorted. On the other hand, $f(\overline{x})$ is of one of the forms 10^t1^{N-t-1} or 0^t1^{N-t} for some $t \ge 0$, since f is nondecreasing. Due to the definition of A, $f(\overline{x})A$ is sorted, a contradiction. □

3. Basic results. In this section, we formally prove the basic results that were introduced in section 1, namely, the existence of efficient worst-case fault-tolerant sorting networks (main theorem) and random-fault-tolerant inserting networks (Theorem 1.2). The proofs are based on Lemma 3.1 and Theorem 1.1, which describe the most difficult parts of the whole construction. Actually, the latter theorem is a special case of the former lemma, but it will be proved separately in section 4, because it is of independent interest and because a solution to the general case will be given on the base of a solution to this special case.

LEMMA 3.1. *There is an explicit construction of a standard N-input* $\mathrm{CORRECT}_k^1$ *network of depth $O(\log N + k)$.*

This is the key lemma in our paper, and we will proceed towards its proof through the rest of the paper; thus the proof is postponed to the end of section 6. In the next lemma we prove that the complement of a standard N-input $\mathrm{CORRECT}_k^1$ network is a $\mathrm{CORRECT}_k^0$ network.

LEMMA 3.2. *There is an explicit construction of a standard N-input* $\mathrm{CORRECT}_k^0$ *network of depth $O(\log N + k)$.*

Proof. Let N and k be fixed positive integers. Let $A = S_1, S_2, \ldots, S_d$, $d = O(\log N + k)$, denote the $\mathrm{CORRECT}_k^1$ network from Lemma 3.1. We would like to prove that the complement of A is a $\mathrm{CORRECT}_k^0$ network. To this end and for the sake of completeness let us recall from [12] the definition and the basic lemma of a network complement.

Let $c(\overline{x})$ denote the complement of 0-1 sequence $\overline{x}$; that is, $c(\overline{x})[i] = 1 - \overline{x}[N-1-i]$ for each $i \in [N]$. For a comparator $[i:j]$ the complement is defined by

$$\overline{[i:j]} = [N-1-j : N-1-i].$$

Clearly, the complement of a standard comparator is standard. Finally, the complement $\overline{A} = \overline{S_1}, \overline{S_2}, \ldots, \overline{S_d}$ of the network $A = S_1, S_2, \ldots, S_d$ is defined by $\overline{S_i} = \{\overline{[i:j]} : [i:j] \in S_i\}$, $i = 1, \ldots, d$.

LEMMA 3.3 (after Schimmler and Starke [12, Lemma 4]). *For every N-input network A and every vector $\overline{x} \in \{0,1\}^N$, $c(c(\overline{x})\overline{A}) = \overline{x}A$.*

Let A be the $\mathrm{CORRECT}_k^1$ network, and let i, j denote nonnegative integers such that $i + j \le k$. In order to prove that $\overline{A}$ is a $\mathrm{CORRECT}_k^0$ network, consider j-fault subnetwork B of $\overline{A}$ and an input $\overline{x}$ obtained from a sorted 0-1 sequence by replacing exactly i zeroes with ones. Then $\overline{B}$ is a j-fault subnetwork of A, and $c(\overline{x})$ is also obtained from a sorted sequence by replacing exactly i ones with zeroes. By the definition of A, $c(\overline{x})\overline{B}$ is sorted. Observe that the complement of a sorted 0-1 sequence is also sorted. Using this and Lemma 3.3, we finally get that $\overline{x}B = c(c(\overline{x})\overline{B})$ is sorted, which completes the proof. □

Using these two lemmas, we are now able to prove our main theorem.

Proof of the main theorem. Let N and k be fixed positive integers. Let S, A, and B

respectively denote the following N-input networks: the AKS sorting network [1], the $\mathrm{CORRECT}_k^1$ network of Lemma 3.1, and the $\mathrm{CORRECT}_k^0$ network of Lemma 3.2. We claim that

$$C = S \cup A^{\Rightarrow depth(S)} \cup B^{\Rightarrow depth(S)+depth(A)}$$

is a k-fault-tolerant sorting network of depth $O(\log N + k)$. Indeed, consider a j-fault subnetwork C' of C; let S', A', and B' denote parts of C' corresponding to S, A, and B with i, j_1, and j_2 faults, respectively, where $i + j_1 + j_2 = j \le k$. If $i = 0$, then C' is a sorting network, because $S' = S$ sorts any input and the following standard networks A' and B' cannot make any change in a sorted sequence.

Now consider $i > 0$, and fix an input 0-1 sequence $\overline{x}$ with l zeroes and $N - l$ ones. Let $\overline{x}' = \overline{x}S'$, $\overline{y} = \overline{x}'A'$, and $\overline{z} = \overline{y}B'$. The proof is completed by showing that $\overline{z}$ is sorted. Due to a lemma of Yao and Yao [13], the Hamming distance between $\overline{x}'$ and $0^l 1^{N-l}$ is at most $2i$; in particular, there are at most i zeroes in the last $N - l$ positions of $\overline{x}'$. As before, denote these by 0^*s. Let $\overline{x}''$ denote the sequence $\overline{x}'$ after replacing all 0^*s with $\frac{1}{2}$, and let $\overline{y}'' = \overline{x}''A'$.

It is well known that for any nondecreasing function f, comparator network A, and input $\overline{x}$, $f(\overline{x})A = f(\overline{x}A)$. We use two such functions $f_1, f_2 : \{0, \frac{1}{2}, 1\} \mapsto \{0, 1\}$ by $f_1(0) = f_2(0) = 0$, $f_1(1) = f_2(1) = 1$, $f_1(\frac{1}{2}) = 0$, and $f_2(\frac{1}{2}) = 1$. Observe that $f_1(\overline{x}'') = \overline{x}'$, $\overline{y} = \overline{x}'A' = f_1(\overline{x}'')A' = f_1(\overline{y}'')$, and $f_2(\overline{x}'')A' = f_2(\overline{y}'')$. The sequence $f_2(\overline{x}'')$ contains only ones in the last $N - l$ positions and at most i ones in the first l positions, and thus it is an appropriate input for a $\mathrm{CORRECT}_k^1$ network. Since $i + j_1 \le k$, Lemma 3.1 implies that $f_2(\overline{y}'')$ is sorted. Furthermore, $\overline{y} = f_1(\overline{y}'')$ differs from this sorted sequence in at most i positions where it has zeroes instead of ones; hence we can apply Lemma 3.2 and finally get that $\overline{z} = \overline{y}B'$ is sorted. □

We conclude this section with a proof of Theorem 1.2. To this end we assume that Theorem 1.1 holds (it is proved in section 4).

Proof of Theorem 1.2. We consider now the case of random faults, where each comparator fails independently with probability upper-bounded by $p < 1$. A random-fault-tolerant INSERT-network should remain an INSERT-network with probability at least $1 - \frac{1}{N}$ in this context. Let us notice that the failure upper bound p can be decreased to p^t by repeating each comparator in a network t times. In this way, we can choose the value of p as small as we need. For a small enough p we prove that the k-fault-tolerant INSERT-network from Theorem 1.1 with $k = O(\log N)$ is a random-fault-tolerant INSERT-network.

Let $s > 0$ be a constant such that $s(\log N + k)$ is an upper bound on the depth of the N-input k-fault-tolerant INSERT-networks. Assuming that $p < \frac{1}{2s}$, let us set $k = \lfloor c \log N \rfloor$, where

$$c = \max\left(2, \frac{16ps}{(1-2ps)^2 \cdot \log e}\right).$$

For the constants selected and a fixed N, let $A = S_1, S_2, \ldots, S_d$, $d \le s(c+1)\log N$, be the $\lfloor c \log N \rfloor$-fault-tolerant INSERT-network. Now apply a random procedure to A, which fails each comparator in A independently with probability less than p, and let A' denote the result. We have to prove that

$$\Pr\{A' \text{ is not an INSERT-network}\} < \frac{1}{N}.$$

Using the insert 0-1 principle, we can restate this inequality as

$$\Pr\{\exists\ 0 < t < N \text{ such that } (10^t 1^{N-t-1})A' \text{ is not sorted}\} < \frac{1}{N}.$$

Hence, it suffices to prove that for any fixed t, $\Pr\{(10^t 1^{N-t-1})A' \text{ is not sorted}\} < N^{-2}$. To this end we apply the principle of deferred decisions; that is, we do not assume that the entire set of random choices is made in advance, but instead we simulate the network A' on $10^t 1^{N-t-1}$, and at the beginning of each stage of A' we fail at random the outgoing comparator of the register in which the first 1 is currently located (other comparators cannot change the contents of registers). Let X_t denote the random variable corresponding to the number of faulty comparators revealed in this procedure. The variable X_t is easily seen to be the sum of at most $s(c+1)\log N$ independent Poisson trials, where each trial has the success probability upper-bounded by p. Thus, a standard Chernoff-type argument [11, Theorem 4.1] implies that for our choice of the constant c

$$\Pr\{X_t > c\log N\} < \frac{1}{N^2}.$$

Since A is a $(c\log N)$-fault-tolerant INSERT-network, we get that the probability that $(10^t 1^{N-t-1})A'$ is not sorted is less than N^{-2}, which completes the proof. □

4. Worst-case fault-tolerant INSERT-network. In this section, we define an N-input INSERT-network that is resistant to k faults and which is of the asymptotically optimal depth $O(\log N + k)$. In this way, Theorem 1.1 will be proved.

The simplest way of inserting an item located in register 0 into a sorted sequence located in registers $1, \ldots, N-1$ is to perform compare-exchange operations on pairs of consecutive registers $[0:1], [1:2], \ldots$. This algorithm is implemented by an odd-even transposition network. The nice property of the algorithm is that it can be easily made fault-tolerant: to tolerate k faults one should add $2k$ stages. The obvious disadvantage is its depth: to move an item from register i to register j one needs $j-i+1$ stages of the network. Since we would like to have a network of a logarithmic depth, we can apply odd-even transposition only at the end of the inserting procedure, when we know that the item is within a logarithmic distance from its final position.

The quickest deterministic way to insert an item is a binary algorithm which is similar to the binary search: we always try to insert the item in the middle of its potential final positions. This algorithm is of logarithmic depth and can be implemented on comparator networks (compare [12]), but it is not easily transformable into an effective fault-tolerant network. The first problem is that we try to move an item from certain positions $\log N$ times; hence a copy of the network could not be applied before time $\log N$. To tolerate k faults we need k copies, and thus the total depth would be of $\Theta(k\log N)$. The second problem is that a simple application of copies is not enough to recover from certain faults: if an item does not move through a faulty comparator it would in the fault-free network, it will probably move through one of the following shorter ones and end up somewhere between its initial and final positions.

In the construction described below, we solve both problems as follows: (1) we apply only one step of the binary insert procedure to an item, and if it does not succeed, we move the item to the next register; (2) we add a set of correcting comparators that recover an item from its wrong positions. Functionally, the network consists of two parts. The first part is intended to do approximate insertion in the presence of at most

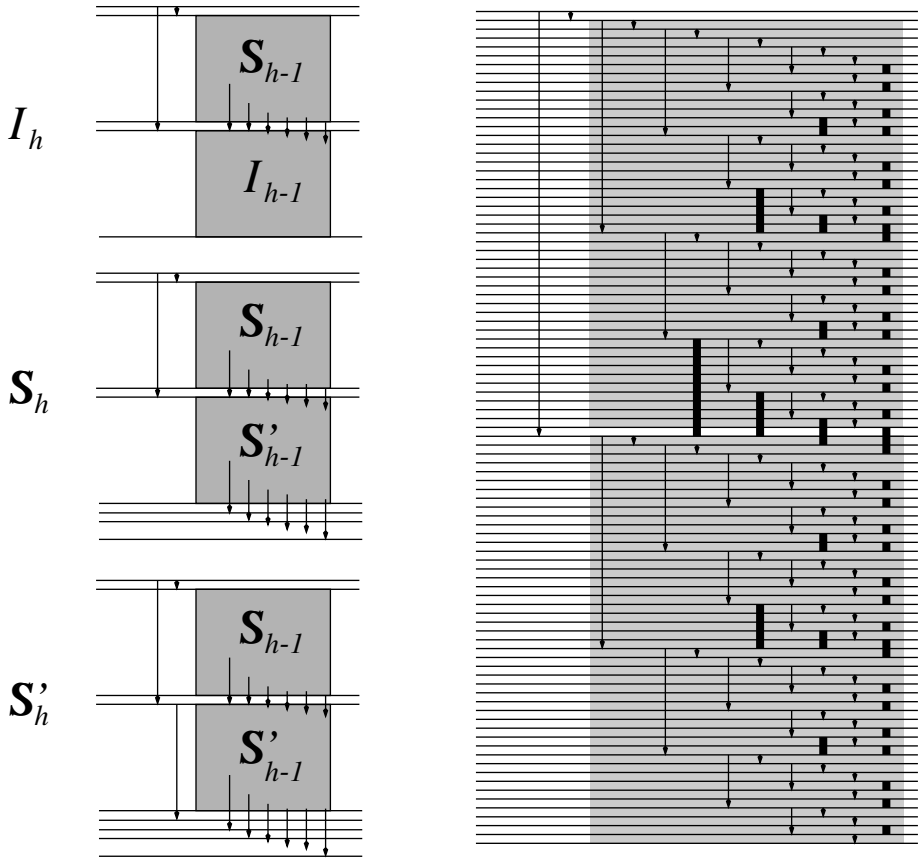


FIG. 4.1. *The recursive structure of I_h is shown on the left-hand side. All visible comparators, except those starting at the register* 0*, are correcting. The I_5 network is depicted on the right-hand side, where the S_4 and I_4 subnetworks are shadowed. All correcting comparators in I_5 are drawn with a thick line.*

k faults, where the end position of an inserted item is at a distance of at most $\log N$ from its correct one. $k+1$ copies of the network I_h (to be described below), each of depth $O(\log N)$ and of a constant delay, form this part. By Proposition 2.1, the total depth of this part is $O(\log N + k)$. The second part is simply the $O(\log N + k)$-depth odd-even transposition network, which adjusts the position of the inserted item to its correct value.

4.1. Definition of inserting networks. The network I_h, $h \geq 0$, contains $N = N_h = 3 \cdot 2^h - 1$ registers and is defined by induction on h with the help of two additional networks S_h and S'_h. In all these networks, we distinguish a certain set of comparators and call them *correcting comparators*. The correcting comparators play a special role in the analysis of the networks. (The name derives from the fact that without correcting comparators, I_h, S_h, and S'_h are all INSERT-networks but do not possess the fault-tolerance property.)

Before we formally define I_h, let us look for a moment at the sample network I_5 in Figure 4.1. It consists of 95 registers numbered $0, 1, \ldots, 94$. Assume that we put the sequence $10^{49}1^{45}$ to them and run a fault-free copy of I_5. It moves the 1 initially located in register 0 into register 49 (it is represented by the second line in the lower

shadowed area) through comparators [0 : 48] and [48 : 49]. But what would happen if [0 : 48] were faulty? Assume now that comparators [0 : 48] and [1 : 25] are faulty and that we have two additional fault-free copies of I_5 after the faulty one. In the faulty I_5, the 1 is moved into register 25 using comparators [0 : 1], [1 : 2], [2 : 14], [14 : 20], [20 : 25]. The last comparator is a correcting one (these are drawn with the thick line), and it enables the next copy of I_5 to recover from the effects of the faulty [1 : 25]. The first fault-free copy moves the 1 further to register 48 using [25 : 37] and another correcting comparator [37 : 48]. Finally, in the second fault-free copy of I_5, comparator [48 : 49] recovers from the effects of the faulty comparator [0 : 48], and we obtain the expected result.

Networks S_h and S'_h contain N_h *normal* registers and some number of *extra* registers denoted by $i_1, i_2, \ldots$. We assume that $N_h \le i_1 \le i_2 \le \cdots$ and that $i_1, i_2, \ldots$ are, in fact, parameters of the networks to be specified later, at the next levels of the recursive definition. In general, there are h extra registers in S_h and $h+1$ in S'_h. The only exception is S_0, where there is a single extra register. Therefore, we will use the notation $S_h(i_1, i_2, \ldots, i_{h'})$ and $S'_h(i_1, i_2, \ldots, i_{h+1})$, where $h' = \max(1, h)$. By $CC(X)$ we will denote the set of correcting comparators of X. The definition of the networks and their sets of correcting comparators follows[3] (see also Figure 4.1):

$$I_0 \stackrel{\text{def}}{=} \{[0:1]\}, \quad S_0(i_1) \stackrel{\text{def}}{=} S'_0(i_1) \stackrel{\text{def}}{=} \{[0:1]\}\{[1:i_1]\},$$

$$CC(I_0) \stackrel{\text{def}}{=} \emptyset, \quad CC(S_0(i_1)) \stackrel{\text{def}}{=} CC(S'_0(i_1)) \stackrel{\text{def}}{=} \{[1:i_1]\}.$$

Let $h \ge 1$, $h'' = \max(h-3, 0)$, and $N' = 3 \cdot 2^{h-1}$. Then

$$\begin{aligned}
I_h &\stackrel{\text{def}}{=} \{[0:N']\}, \{[0:1]\} \cup (I_{h-1}^{\Rightarrow 2} \Downarrow N') \\
&\quad \cup (S_{h-1}^{\Rightarrow 2}(N', N', N'+1, N'+2, \ldots, N'+h'') \Downarrow 1), \\
S_h(i_1, \ldots, i_h) &\stackrel{\text{def}}{=} \{[0:N']\}, \{[0:1]\} \cup (S'^{\Rightarrow 2}_{h-1}(i_1, \ldots, i_h) \Downarrow N') \\
&\quad \cup (S_{h-1}^{\Rightarrow 2}(N', N', N'+1, N'+2, \ldots, N'+h'') \Downarrow 1), \\
S'_h(i_1, \ldots, i_{h+1}) &\stackrel{\text{def}}{=} \{[0:N']\}, \{[0:1][N':i_1]\} \cup (S'^{\Rightarrow 2}_{h-1}(i_2, \ldots, i_{h+1}) \Downarrow N') \\
&\quad \cup (S_{h-1}^{\Rightarrow 2}(N', N', N'+1, N'+2, \ldots, N'+h'') \Downarrow 1).
\end{aligned}$$

The rule for finding correcting comparators in the networks I_h, $S_h(i_1, \ldots, i_{h'})$, and $S'_h(i_1, \ldots, i_{h+1})$ is that, whenever a comparator is defined to be correcting, it remains such at the next steps of the recursive definition. Therefore, we should only say which of the comparators introduced at the step h is the correcting one. In fact, there is only one new correcting comparator, namely $[N' : i_1]$, in the definition of $S'_h(i_1, \ldots, i_{h+1})$. Formally, the sets are defined by the following equations:

$$\begin{aligned}
CC(I_h) &\stackrel{\text{def}}{=} CC(I_{h-1}^{\Rightarrow 2} \Downarrow N') \\
&\quad \cup CC(S_{h-1}^{\Rightarrow 2}(N', N', N'+1, N'+2, \ldots, N'+h'') \Downarrow 1), \\
CC(S_h(i_1, \ldots, i_h)) &\stackrel{\text{def}}{=} CC(S'^{\Rightarrow 2}_{h-1}(i_1, \ldots, i_h) \Downarrow N') \\
&\quad \cup CC(S_{h-1}^{\Rightarrow 2}(N', N', N'+1, N'+2, \ldots, N'+h'') \Downarrow 1), \\
CC(S'_h(i_1, \ldots, i_{h+1})) &\stackrel{\text{def}}{=} \{[N' : i_1]\} \cup CC(S'^{\Rightarrow 2}_{h-1}(i_2, \ldots, i_{h+1}) \Downarrow N') \\
&\quad \cup CC(S_{h-1}^{\Rightarrow 2}(N', N', N'+1, N'+2, \ldots, N'+h'') \Downarrow 1),
\end{aligned}$$

[3] In the definition we assume that the shift down operation $\Downarrow$ is applied only to normal registers; the transformation of extra registers is given explicitly by assigning new values to parameters.

where the shift operations preserve the property of being a correcting comparator.

Notice that when we delete extra registers in a network $S(i_1, \ldots, i_{h'})$ (or in $S'_h(i_1, \ldots, i_{h+1})$) and comparators that use them, then the network becomes identical to I_h.

It is not quite clear whether the stages of the networks defined above consist of disjoint comparators only. Namely, at the stage h we assign extra registers of S_{h-1} to normal registers, and conflicts could occur between comparators that used those extra registers and comparators that used the normal registers to which they are assigned. Lemma 4.1 states that this is not the case.

LEMMA 4.1. *Let $h \geq 0$, $h' = \max(1, h)$, and X_h denote one of the networks I_h, $S_h(i_1, \ldots, i_{h'})$, or $S'_h(i_1, \ldots, i_{h+1})$. Then all comparators in any stage of X_h are pairwise disjoint and delay$(X_h) \leq 8$.*

Proof. Network X_0 is explicitly given, so assume $h \geq 1$. Register 0 is connected to two comparators at stages 1 and 2. Hence $fst(0, X_h) = 1$ and $lst(0, X_h) = 2$. Using induction on h, we shall prove the following facts about X_h, $h \geq 1$:

(a) $fst(j, X_h) \geq 2j$ and $lst(j, X_h) \leq 2j + 2$ for $1 \leq j \leq h$;
(b) $fst(i_j, S'_h) = lst(i_j, S'_h) = 2j$ for $j = 1, 2, \ldots, h + 1$;
(c) $fst(i_j, S_h) = lst(i_j, S_h) = 2j + 2$ for $j = 1, 2, \ldots, h$;
(d) $lst(j, X_h) - fst(j, X_h) \leq 7$ for $h < j < N_h$;
(e) at each stage of X_h a register j is involved in at most one comparison, $j = 0, \ldots, N_h - 1, i_1, i_2, \ldots$.

For $h = 1$ these properties could be easily checked by drawing the simple networks. Assume $h \geq 2$ and that the facts hold for $h - 1$. Let $N'_h = 3 \cdot 2^{h-1}$.

(a) Look at the layout of comparators in Figure 4.1. Register 1 is compared directly at stage 2, and it is used as register 0 by S_{h-1}; thus it is compared also at stages 3 and 4. It follows that $fst(1, X_h) = 2$ and $lst(0, X_h) = 4$. Registers $2, \ldots, h$ are used by the subnetwork $(S^{\Rightarrow 2}_{h-1}) \Downarrow 1$ only, where they correspond to registers $1, \ldots, h-1$ of S_{h-1}. By the induction hypothesis, $fst(j, S_{h-1}) \geq 2j$ and $lst(j, S_{h-1}) \leq 2j + 2$ for $1 \leq j \leq h - 1$. Stages of S_{h-1} are shifted two positions to the right in X_h, and thus $fst(j, X_h) \geq 2j$ and $lst(j, X_h) \leq 2j + 2$ for $2 \leq j \leq h$.

(b) Extra register i_1 of S'_h is used only once at stage 2. Thus $fst(i_1, S'_h) = lst(i_1, S'_h) = 2$. Extra registers $i_2, \ldots, i_{h+1}$ of S'_h are mapped to $i_1, \ldots, i_h$ of $(S'^{\Rightarrow 2}_{h-1}) \Downarrow N'_h$. By induction, $fst(i_j, S'_h) = lst(i_j, S'_h) = 2j$ for $j = 1, 2, \ldots, h$. Since all stages of S'_{h-1} are shifted two positions to the right in X_h and the index j is increased by 1, fact (b) holds also for $i_2, \ldots, i_h + 1$.

(c) Extra registers $i_1, \ldots, i_h$ of S_h are mapped to extra registers $i_1, \ldots, i_h$ of $(S'^{\Rightarrow 2}_{h-1}) \Downarrow N'_h$. Using fact (b) and inductive arguments similar to those above, we get that fact (c) is also true.

(d) Consider the set of registers $\{h + 1, \ldots, N_h - 1\}$. Only registers $N'_h, \ldots, N'_h + \max(h - 3, 0)$ are involved in new comparisons at the level h of the recursive definition of X_h. For the others the inductive hypothesis applies. Registers $N'_h + j$, $j = 0, \ldots, \max(h - 3, 0)$, are assigned to the extra registers of $(S^{\Rightarrow 2}_{h-1}) \Downarrow 1$, which are used at stages $6, 8, \ldots, 2h+2$, respectively, by fact (c). (We have taken into account the shift to the right.) On the other hand, registers $N'_h, \ldots, N'_h + \max(h-3, 0)$ correspond to the initial registers of the subnetwork I_{h-1} or S'_{h-1}, and thus, by fact (a), they are used only at stages $2j+2$, $2j+3$, $2j+4$ (with the shift). Combining these, we conclude that register N'_h is used only at stages $1, 2, 3, 4, 6, 8$ (notice that the first comparator also uses it, and it is assigned to i_1 and i_2), and register $N'_h + j$, $1 \leq j \leq \max(h-3, 0)$, is used at stages $2j + 2$, $2j + 3$, $2j + 4$, $2j + 8$. This proves the claim.

(e) It follows from the considerations in (a)–(d) and the induction hypothesis that each register is used at most once in any stage of X_h. □

COROLLARY 4.2. *If $[i:j]$ is a correcting comparator at stage l of X_h, then there does not exist $[j:i']$ for any i' at any stage later than l in X_h. Moreover, if $[i:j]$ is not a correcting comparator at stage l of X_h, then there does not exist $[j:i']$ for any i' at any stage earlier than l in X_h.*

Proof. Consider the stage h of the construction of X_h in the inductive proof above. New correcting comparators of the form $[* : j]$, where j is an index of a normal register, could appear only for $j \in \{N', N'+1, \ldots, N'+h-1\}$. But it is easy to notice that such comparators are the last ones in the sequence of comparisons in which j is involved. To prove the second part of the corollary, observe that (1) j must be greater than zero, (2) new comparators of the form $[j : *]$ are added only to the first two stages of X_h, and (3) the new comparators are outgoing comparators from registers 0 or N'. By (1) and (3), we should consider only $j = N'$, and, by (2), there is only one new comparator of the form $[j : *]$, namely $[N' : i_1]$ at stage 2. The only noncorrecting comparator $[i : N']$ for any i is added in stage 1, thus earlier than stage 2. □

We shall use the insert 0-1 principle to prove that, in the presence of at most k faulty comparators, $k+1$ copies of I_h approximately insert an item initially located in register 0 into a nondecreasing sequence of items initially located in registers $1, \ldots, N_h - 1$. Therefore, we will consider only inputs of the form 10^t1^{N-t-1}, where $1 \le t \le N-1$. To distinguish the 1 that should be inserted, we will denote it by 1^*.

For a given input 10^t1^{N-t-1} with t zeroes we will say that a comparator $[i:j]$ is *active* (with respect to t) if $j \le t$. Otherwise it is *inactive*. Thus, an inactive comparator (with respect to t) in a standard network never moves any value, when 10^t1^{N-t-1} is initially put to its registers. In the fault-free case, one copy of I_h is enough to move 1^* to register t. On the other hand, if there are faults in I_h, we need additional copies of I_h to recover from a wrong register to which 1^* could be moved. In section 4.2, based upon the structure of active comparators in I_h, we will assign weights to registers in I_h to measure the progress in the recovery procedure from the effect of faulty comparators, which 1^* has already met.

From now on, by X_h we will denote one of the networks I_h, $S_h(i_1, \ldots, i_h)$, or $S'_h(i_1, \ldots, i_{h+1})$. Consider the sequence $([i:j_0], [i:j_1], \ldots, [i:j_{m_i}])$ of all outgoing comparators from a register i, $0 \le i \le N-1$, in a network X_h, where the order of comparators is the same as the order of stages, to which the stages belong.

FACT 1. *The sequence of all outgoing comparators from a register i, $0 \le i \le N-1$, in X_h is empty or in one of the following forms:*

(a) $([i:l], [i:j_1], [i:j_2])$,
(b) $([i:l], [i:j_1])$,
(c) $([i:l])$,
(d) $([i:j_0], [i:j_1])$,
(e) $([i:j_0])$,

where $[i:l] \in CC(X_h)$, $[i:j_0]$, $[i:j_1]$, $[i:j_2] \notin CC(X_h)$, and $i < j_2 < j_1 < l$ or $i < j_1 < j_0$.

Remark. Let $10^t1^{N_h-t-1}$ be an input to I_h, and consider the sequence of active comparators (with respect to t) outgoing from a register i, $0 \le i < N_h$. It should be noticed that each such sequence also fulfills the statement of Fact 1. There are two reasons for that: (1) active comparators form an end segment of a sequence of all outgoing comparators, since they are shorter than inactive ones, and (2) sequences in the cases (a)–(e) represent also all their end segments.

4.2. Register weights. Assume that 1^* is now in a register i and that there are active comparators outgoing from i. In such a situation, 1^* will move through the first active comparator that is not faulty. The weight $c_h(i,t)$, defined formally below, is supposed to estimate the number of fault-free copies of I_h that are needed to move 1^* from the register i close to the register t. The definition depends only on the structure of I_h (fault-free). However, the definition is formulated in such a way that it will be used to bound the effects of "wrong" movements of 1^* due to faulty comparators. In other words, if we consider the sequence of faulty subnetworks of I_h, then this number will also describe the difference between the number of faulty comparators that have blocked 1^* on the way from the initial register to the register i and the number of correcting comparators through which 1^* has moved. Roughly speaking, each faulty comparator increases the weight by 1, and each correcting comparator decreases it by 1.

The outline of technical details in this subsection is as follows. In Lemma 4.3, we consider a single copy of I_h with k faults and give an upper bound on the weight difference between the output and input registers of 1^*. In Lemma 4.4, we analyze the number of copies that are required to reduce the weight of the output register to zero when k is the upper bound on the total number of faults in all copies. Next, in Fact 2 we prove that registers of weights zero are within logarithmic distance from the expected final position of 1^*. Finally, in Lemma 4.5 we estimate the weight of register 0 and the weights of all other registers for all inputs under consideration. We finish this subsection by proving Theorem 1.1.

Let $h \geq 0$ and $0 \leq t \leq N_h - 1$. Inductively, for $i = N_h - 1, N_h - 2, \ldots, 0$ we define the weight $c_h(i,t)$ of register i in I_h with respect to the goal register t. If $i > t$ or the sequence of active comparators outgoing from i is empty, then $c_h(i,t) \stackrel{\text{def}}{=} 0$. Otherwise, the sequence has to have one of the forms described in Fact 1. Using the notation from Fact 1, we define

$$c_h(i,t) \stackrel{\text{def}}{=} \max(1, c_h(l,t)+1, c_h(j_0,t), c_h(j_1,t)-1, c_h(j_2,t)-2), \tag{4.1}$$

where any terms that are undefined for a particular case should be replaced by 0.

The basic properties of weights $c_h(i,t)$ with respect to the behavior of a faulty version of I_h are summarized in the following two lemmas.

LEMMA 4.3. *Let $h \geq 0$, $N_h = 3 \cdot 2^h - 1$, and $0 \leq i < t \leq N_h - 1$. Let I'_h denote any k-fault subnetwork of I_h, and let $\overline{x_i} = 0^i 10^{t-i} 1^{N-t-1}$ be an input to I'_h. Then the output $\overline{x_i} I'_h$ has a form $\overline{x_j} = 0^j 10^{t-j} 1^{N-t-1}$, and if $c_h(j,t) \neq 0$, then*

$$c_h(j,t) - c_h(i,t) \leq k - 1.$$

Proof. The network I'_h is standard, because all comparators in I_h are standard. Therefore, the only 1 that can possibly move is the 1 initially located in the register i (referred to as 1^* in the following). Thus the output takes the form $\overline{x_j}$ for some $j \geq i$.

Since $c_h(j,t) \neq 0$ is assumed, the condition $c_h(i,t) \geq 1$ must be true (otherwise, since $i < t$ and $c_h(i,t) = 0$, there is no active comparator outgoing from i, and so $j = i$, a contradiction). Consider the sequence $i = i_0 < i_1 < \cdots < i_s = j$, $s \geq 0$, such that 1^* moves through comparators $[i_0 : i_1], [i_1 : i_2], \ldots, [i_{s-1} : i_s]$ in a given order (in the faulty network I'_h). Due to Corollary 4.2, only the last comparator could be a correcting one, and if $[i_{m-1} : i_m]$, $0 < m \leq s$, is a noncorrecting comparator at stage l_m, then all comparators outgoing from i_m are at stages later than l_m. Consider now all but the last comparators in the sequence.

By Fact 1, we know the structure of the sequence of comparators outgoing from register i_m, $0 \le m < s$. At the beginning of the sequence there could be inactive comparators, followed by zero or more faulty comparators, and then there is the comparator $[i_m : i_{m+1}]$. Let k_m denote the number of active faulty comparators outgoing from i_m before $[i_m : i_{m+1}]$. Then, by the definition of $c_h(i_m, t)$,

$$c_h(i_m, t) \ge c_h(i_{m+1}, t) - k_m. \tag{4.2}$$

For the last comparator in the sequence, we have two cases, as follows.

Case $[i_{s-1} : i_s] \in CC(I_h)$. In this case, by Fact 1, k_{s-1} must be zero, and from (4.1) we can get a stronger inequality $c_h(i_{s-1}, t) \ge c_h(i_s, t) + 1$. Summing up the inequality with (4.2) over all $m = 0, \ldots, s-2$, we get

$$c_h(i_s, t) - c_h(i_0, t) \le k_0 + k_1 + \cdots + k_{s-2} - 1 \le k - 1.$$

Case $[i_{s-1} : i_s] \notin CC(I_h)$. Because $c_h(i_s, t) > 0$, there is at least one active comparator outgoing from i_s. By Corollary 4.2, comparators outgoing from i_s must occur at stages later than that of $[i_{s-1} : i_s]$. Due to Fact 1 and our definition of i_s, all active comparators outgoing from i_s must be faulty. Therefore $k_0 + k_1 + \cdots + k_{s-1} \le k - 1$. Summing up the inequalities as in the previous case, we get the desired result. □

LEMMA 4.4. *Let* $L_h^s \stackrel{\text{def}}{=} \{i \in [N_h] : \forall\ i \le t \le N_h - 1,\ c_h(i, t) \le s\}$, *where* $h \ge 0$, $s > 0$, *and* $N_h = 3 \cdot 2^h - 1$. *Moreover, let* $k \ge k' \ge 0$ *and* A *be any* k'*-fault subnetwork of* $I_h^{(k+s)}$. *Then for an input* $\overline{x_i} = 0^i 10^{t-i} 1^{N_h - t - 1}$, $0 \le i \le t \le N_h - 1$ *and* $i \in L_h^s$, *the output* $\overline{x_i}A = 0^j 10^{t-j} 1^{N_h - t - 1}$ *has the property that* $c_h(j, t) = 0$.

Proof. Assume to the contrary that there are h, s, k, k', A, i, t described in the lemma such that $c_h(j, t) > 0$. Let $k_1, k_2, \ldots, k_{k+s}$ denote the number of faulty comparators in the first, the second, ..., the $(k+s)$th copy of I_h. Let $i = i_0 \le i_1 \le \cdots \le i_{k+s} = j$ denote the positions of the first 1 between the corresponding copies. Clearly, $c_h(i_m, t) > 0$ for $0 \le m \le k + s$. Using Lemma 4.3 for each k_m-fault subnetwork of I_h and summing up the inequalities, we get

$$c_h(i_{k+s}, t) - c_h(i_0, t) \le \left(\sum_{m=1}^{k+s} k_m \right) - (k + s).$$

Because $\sum_{m=1}^{k+s} k_m \le k' \le k$ and $c_h(i, t) \le s$, we have arrived at the contradiction $c_h(j, t) \le 0$. □

One can observe that 1^* has no chance to move when it is in a register i such that $c_h(i, t) = 0$. Fortunately, in such cases 1^* is very close to its target position t.

FACT 2. *Let* $h \ge 0$ *and* $0 \le t \le N_h - 1$. *Consider a register* i, $i < t$, *such that* $c_h(i, t) = 0$. *Then* $t - i \le h$.

Proof. By induction on h, we prove that for each register i, $0 \le i \le N_h - 2$, in X_h at least one comparator outgoing from i is of length $\le h + 1$.

Basis: $h = 0$. There is only one comparator $[0 : 1]$ outgoing from 0.

Inductive step: $h > 0$. Let $N' = 3 \cdot 2^{h-1}$. For $1 \le i \le N' - 2$ and $N' \le i \le N_h - 2$ the statement holds due to the inductive assumption about I_{h-1}, S_{h-1}, and S'_{h-1}. Therefore, we should consider only registers 0 and $N' - 1$. By construction, there is a comparator of length 1 outgoing from 0. In case of $N' - 1$, there is a correcting comparator $[N' - 1 : N' + \max(h - 3, 0)]$ in X_h of length not greater than $h + 1$, and we are done.

Assume now that $c_h(i,t) = 0$ for some $i < t$. Then each comparator $[i : j]$ is inactive, particularly those of length $\leq h+1$. Because $j \geq t+1$, $t-i \leq h$ is true and Fact 2 holds. □

To prove Theorem 1.1, we need only to estimate the weight $c_h(i,t)$ for i in particular sets of register indexes. In fact, we are interested in two sets: $\{0\}$ and $\{0,1,\ldots,N-1\}$.

LEMMA 4.5. *Let $h \geq 0$ and $0 \leq t \leq N_h - 1$. Then*

(i) $c_h(0,t) \leq 1$,

(ii) $0 \leq c_h(i,t) \leq h+1$ *for* $0 \leq i \leq t$.

We postpone the proof of Lemma 4.5 to the next subsection. Now, we can prove the main result of this section, namely, Theorem 1.1.

Proof of Theorem 1.1. We should prove that for each $N > 0$ and $k \geq 0$ there exists an explicit construction of an N-input k-fault-tolerant INSERT-network of depth $O(\log N + k)$.

Assume $N > 2$. (For $N = 2$ one takes $(k+1)$-times the comparator $[0 : 1]$.) Let $h \geq 1$ be such that $3 \cdot 2^{h-1} - 1 < N \leq 3 \cdot 2^h - 1$. Consider the network A that consists of $I_h^{(k+1)}$ followed by $h + 2k$ stages of the odd-even transposition network. A contains $N_h = 3 \cdot 2^h - 1 \geq N$ registers. To get the N-input network, we delete the last superfluous registers and comparators that use them. Let the truncated network be called B. We prove that B has the desired properties.

Since $depth(I_h) = 2h + 2$, $depth(I_h^{(k+1)}) = 2h + 2 + 8k$, with respect to Proposition 2.1 and Lemma 4.1. Accordingly, $depth(B) \leq depth(A) = 3h + 10k + 3$. Since $h = \lceil \log \frac{N+1}{3} \rceil$, the total depth of B is of $O(\log N + k)$.

To prove that B is a k-fault-tolerant INSERT-network, we will use the insert 0-1 principle. Therefore, we consider only the behavior of B on inputs of the form $\overline{y_t} \stackrel{\text{def}}{=} 10^t 1^{N-t-1}$, where $0 \leq t \leq N-1$. However, the behavior of B is the same as the behavior of the first N registers of A on the corresponding input $\overline{y'_t} \stackrel{\text{def}}{=} 10^t 1^{N_h-t-1}$, because the deleted registers contain 1's in A, and thus the deleted comparators are inactive.

By Lemma 4.5, $c_h(0,t) \leq 1$. Using Lemma 4.4 with $s = 1$ and $L_h^s \supseteq \{0\}$, we get that after $I_h^{(k+1)}$, item 1^* is in a register j such that $c_h(j,t) = 0$. According to Fact 2, $t - j \leq h$. By standard arguments, $h + 2k$ stages of an odd-even transposition will move 1^* to the register t, even in the presence of k faulty comparators. That proves that A is a k-fault-tolerant INSERT-network, and, by the arguments above, so is B. □

Remark. Due to Lemma 4.5(ii), the item to be inserted can be given in any register (i.e., not necessarily in register 0), and such a generalized INSERT-network is still of depth $O(\log N + k)$. To construct the network we should take $(h + k)$ copies of I_h, followed by $(h + k)$ copies of the symmetric version of I_h (replace each comparator $[i : j]$ in I_h by $[N - j : N - i]$), and then the odd-even transposition part. The same arguments as above prove that the network is correct.

4.3. Estimation of weights. This subsection is devoted to the proof of Lemma 4.5, that is, to an estimation of the values $c_h(i,t)$. Since the weights are defined in a sequential manner and the structure of I_h is described recursively, we need a tool that allows us to apply induction on the structure of I_h in order to get the estimation. The weights below are defined recursively based on the structure of the networks, but they possess sequential properties similar to those of the weights $c_h(i,t)$ and, in consequence, will be used as their upper bounds.

Weights $w_h(i)$ for registers in $S_h(i_1, \ldots, i_h)$ (and similar weights $w'_h(i)$ for registers in $S'_h(i_1, \ldots, i_{h+1})$) are defined recursively below. Let $h > 0$, $N_h = 3 \cdot 2^h - 1$, and $N'_h = 3 \cdot 2^{h-1}$. Then

$$w_0(j) \stackrel{\text{def}}{=} w'_0(j) \stackrel{\text{def}}{=} \begin{cases} 2, & j = 0, 1, \\ 1, & j = i_1, \end{cases}$$

$$w_h(j) \stackrel{\text{def}}{=} \begin{cases} 2, & j = 0, \\ w_{h-1}(j-1) + 1, & 0 < j < N'_h, \\ w'_{h-1}(j - N'_h), & N'_h \le j < N_h, \\ l, & j = i_l,\ l \le h, \end{cases}$$

$$w'_h(j) \stackrel{\text{def}}{=} \begin{cases} 2, & j \in \{0, N'_h\}, \\ w_{h-1}(j-1) + 1, & 0 < j < N'_h, \\ w'_{h-1}(j - N'_h) + 1, & N'_h < j < N_h, \\ l, & j = i_l,\ l \le h+1. \end{cases}$$

We will need the following properties of the weights.

FACT 3. *Let $h \ge 0$. Then $w_h(j) = w'_h(j) = j + 2$ for $j = 0, 1, \ldots, h$. Moreover, $w_h(j) \le h + 2$ and $w'_h(j) \le h + 2$ for $j = 0, 1, \ldots, N_h - 1$.*

Proof. The proof follows from an easy induction. □

FACT 4. *Let $h \ge 0$ and $0 \le i \le N_h - 1$. Consider the sequence of comparators in S_h (or S'_h) outgoing from the register i, which must be of a form described in Fact* 1. *Then, using the notation given there,*

$$w_h(i) \ge \max(w_h(l) + 1, w_h(j_0), w_h(j_1) - 1, w_h(j_2) - 2),$$

$$w'_h(i) \ge \max(w'_h(l) + 1, w'_h(j_0), w'_h(j_1) - 1, w'_h(j_2) - 2),$$

where any terms that are undefined for a particular case should be replaced by zero.

Proof. The idea of the proof is to proceed by induction and reduce the inequalities for h to the corresponding inequalities for $h - 1$. Separately, we should check only registers that have new outgoing comparators, that is, the register 0 in S_h and the registers $0, N'_h$ in S'_h. Moreover, it should be verified that, in the process of assigning extra registers from the level $h - 1$ to registers from the level h, their weights do not increase more than the recurrence allows (otherwise, we could not apply induction). We check only the register 0 in S_h, the register N'_h in S'_h, and the reassignment of extra registers in the definition of S'_h. The other cases are left to the reader.

Case 0 *in* S_h. The sequence of outgoing registers from 0 in S_h is $([0 : N'_h], [0 : 1])$. Since $w_h(0) = 2$, $w_h(N'_h) = w'_{h-1}(0) = 2$, and $w_h(1) = w_{h-1}(0) + 1 = 3$, the inequality $w_h(0) \ge \max(w_h(N'_h), w_h(1) - 1)$ holds.

Case N'_h *in* S'_h. The corresponding sequence for N'_h in S'_h is $([N'_h : i_1], [N'_h : N'_h + N'_{h-1}], [N'_h : N'_h + 1])$, where the first comparator is a correcting one and the last one is absent for $h = 1$. By induction, we know that $2 = w'_{h-1}(0) \ge \max(w'_{h-1}(N'_{h-1}), w'_{h-1}(1) - 1)$. Since $w'_h(N'_h) = 2$, $w'_h(i_1) = 1$, $w'_h(N'_h + N'_{h-1}) = w'_{h-1}(N'_{h-1}) + 1 \le 3$, and $w'_h(N'_h + 1) = w'_{h-1}(1) + 1 \le 4$, the inequality $w'_h(N'_h) \ge \max(w'_h(i_1) + 1, w'_h(N'_h + N'_{h-1}) - 1, w'_h(N'_h + 1) - 2)$ holds.

Case extra registers in S'_h. For each register j in S'_h, $N'_h < j < N_h$, we have $w'_h(j) = w'_{h-1}(j - N'_h) + 1$. Hence, the weights increase by 1 in this part of the network. We should check that the weights of extra registers in S'_{h-1} do not increase more that 1 through the reassignment in S'_h. However, a register i_l, $1 \le l \le h$, in S'_{h-1} is assigned to i_{l+1} in S'_h, and therefore, its weight increases by exactly 1.

Let $h' = \max(h-1, 1)$. Consider the subnetwork S_{h-1} in S'_h. Its extra registers $i_1, \ldots, i_{h'}$ have weights $w_{h-1}(i_1) = 1, \ldots, w_{h-1}(i_{h'}) = h'$ and are assigned to normal registers $N', N', N'+1, \ldots, N'+h'-2$. Since $w'_h(N') = 2$ and $w'_h(N'+j) = w'_{h-1}(j)+1 = j+3$ for $j = 1, \ldots, h'-2$ (the last equality being due to Fact 3), the weights of the extra registers have increased by at most 1. However, $w'_h(j)$ increases by exactly 1 with respect to $w_{h-1}(j-1)$, where $0 < j < N'_h$. Consequently, we can use induction to prove Fact 4 for all other registers in S'_h. □

Proof of Lemma 4.5. We shall prove the lemma by induction on h. To this end we extend (i) to (i′) and add a new statement (iii).

(i′) $c_h(i,t) \le i+1$ for $0 \le i \le \min(h,t)$,

(ii) $0 \le c_h(i,t) \le h+1$ for $0 \le i \le t$,

(iii) $c_h(i,t) \le w_{h-1}(i-1)$ for $h \ge 1$ and $1 \le i < 3 \cdot 2^{h-1} \le t$,

where weights $w_h(i)$ for registers in $S_h(i_1, \ldots, i_h)$ have been defined above. In the following, induction is made on h for all the three inequalities being proved.

Directly from the definition, one can check that (i′), (ii), and (iii) hold for $h = 0, 1$. Assume that $h > 1$. Let $N_h = 3 \cdot 2^h - 1$ and $N'_h = 3 \cdot 2^{h-1}$. In the proof, we will use the following observation concerning weights $c_h(i,t)$ in I_h:

$$c_h(i,t) = \begin{cases} c_{h-1}(i-1, t-1), & 1 \le i \le t < N'_h, \\ c_{h-1}(i-N'_h, t-N'_h), & N'_h \le i \le t < N_h. \end{cases}$$

The first equality is a consequence of the fact that if $1 \le t < N'_h$, then all registers assigned to extra registers of S_{h-1} contain 1's, and therefore, the subnetworks of S_{h-1} and I_{h-1} consisting of active comparators only (with respect to $t-1$) are identical. Recall that I_h and S_h are identical after deleting extra registers in S_h and the comparators that use them. The second equality follows directly from the definitions.

First, we prove (iii) for a fixed t, proceeding by induction backwards on i and using Fact 4. Assume first that $t \ge N'_h + h - 3$. Then for each register i, $1 \le i < N'_h$, all its outgoing comparators are active. Therefore, the symbols l, j_0, j_1, and j_2 in the definition of $c_h(i,t)$ and in the inequalities of Fact 4 denote the right-hand ends of the same comparators. If, by induction, (iii) holds for l, j_0, j_1, and j_2, then it holds for i too. To start the induction, we should check only whether (iii) holds for registers assigned to the extra registers of S_{h-1}. A register i_m, $1 \le m < h$, has the weight $w_{h-1}(i_m) = m$ and is assigned to the register $j_m = N'_h + \max(0, m-2)$ with weight $c_h(j_m, t) = c_{h-1}(j_m - N'_h, t - N'_h) \le \max(1, m-1) \le m$, where we use (i′) from the inductive assumption about I_{h-1}. By induction, (iii) follows.

If $N'_h \le t < N'_h + h - 3$, then there are inactive correcting comparators outgoing from registers in $\{1, \ldots, N'_h - 1\}$. It is not difficult to see that these comparators are exactly $[N'_h - 1 : N'_h + h - 3], [N'_h - N_0 : N'_h + h - 4], [N'_h - N_1 : N'_h + h - 5], \ldots, [N'_h - N_{m_t} : t+1]$, where $m_t = N'_h + h - t - 5$. Moreover, the structure of active outgoing comparators for a register i in an interval $[N'_h - N_j, N'_h - 1]$, $j \le m_t$, is the same as the structure of all comparators outgoing from i in I_j. Hence, $c_h(N'_h - N_j, t) = c_j(0, N_j - 1) \le 1$. Since $w_{h-1}(j) \ge 2$ is true for any j, the inequality (iii) holds for a register i in the set $\{N'_h - N_j : j \le m_t\}$. For a register not in the set, all its outgoing comparators are active, and the inductive arguments from the previous paragraph are valid.

(ii) Obviously, $c_h(0,0) = 0$. Assume $t > 0$. If $t < N'_h$, then $c_h(i,t) = c_{h-1}(i-1, t-1) \le h$ by induction for all i, $1 \le i \le t$. Since $[0 : N'_h]$ is inactive, $c_h(0,t) = c_h(1,t) \le h$. If $N'_h \le t < N_h$, then $c_h(i,t) = c_{h-1}(i - N'_h, t - N'_h) \le h$ for $N'_h \le i \le t$. For other values of i we use (iii) and Fact 3. Namely, if $1 \le i < N'_h$, then $c_h(i,t) \le$

$w_{h-1}(i-1) \leq h+1$. By definition and the inequalities above, $c_h(0,t)$ is also bounded by $h+1$.

(i′) By similar arguments as in (ii), one can get the inequality. □

5. Bucket fault-tolerant INSERT-network. In this section, we describe the second part of the CORRECT$_k^1$ network, namely, bucket-insertion of 1*s. The problem we will deal with is quite similar to the problem of inserting an item into a sorted sequence (discussed in the previous section), but instead of one displaced item, there are s, $s \leq k$, such items. As before, we will use a generalized insert 0-1 principle; that is, we assume that an input to our network consists of a sequence of 0's with at most k "displaced" 1*s spread inside it, followed by a sequence of 1's.

As we showed in the previous section, in case of one 1^* to be inserted, there is a well-defined path in the INSERT-network, along which the item can be moved to its destination, and only faulty comparators on the path can make the item to switch to another (wrong) path. In case there are more 1*s to be inserted, a comparison between two 1*s causes the upper one to be blocked on its path by the lower 1^*. We call such a situation a *conflict*. For the upper 1^* the effect of a conflict is the same as if it met a faulty comparator. Potentially, we can have up to $k/2$ conflicts at each stage of the INSERT-network. To avoid this, we group consecutive k registers into so-called *buckets* and organize the movement of items between buckets in such a way that conflicts occur inside buckets only.

The idea is to replace each register in the INSERT-network with a bucket of registers, and each comparator $[i : j]$ with the comparator $[l_i : u_j]$, where l_i is the lower (last) register in the ith bucket and u_j is the upper (first) register in the jth bucket. To make the construction work, subnetworks based on a MINMAX-network are put in each bucket so that the MAX-item in one bucket is compared to the MIN-item in the other bucket. Since the buckets are of size k, this will never cause a situation where 1^* in the MAX-register of a bucket is compared to 1^* in the MIN-register of another one. Thus, the conflicts outside buckets will be avoided unless there are faulty comparators.

We start by fixing some definitions. Let $k > 1$ be an integer. We say that a bucket of size k is *empty* (respectively, *full*) if it contains k zeroes (respectively, k ones). The register $l_i = i \cdot k$ and $u_i = (i+1) \cdot k - 1$ will be called the *MIN-register* and the *MAX-register*, respectively, of the ith bucket.

Constructing the networks for buckets, we would like to achieve the following three goals in the fault-free case:

1. An incoming 1 is never blocked by another 1 in the MIN-register of a bucket.
2. If a bucket is nonempty, a 1 is moved into the MAX-register before its first outgoing comparator is used.
3. If a bucket is empty and it has an incoming comparator preceding the first outgoing one, the incoming 1 is moved directly to the MAX-register before the first outgoing comparator is used.

If the network contains faulty comparators and some of the goals are not achieved, we would like to assign a distinct faulty comparator to each such a goal. (Each incoming 1 that has been blocked counts on its own.)

Recall from the previous section that the INSERT-network consists of $k+1$ copies of I_h followed by $h+2k$ stages of an odd-even transposition network (cf. the proof of Theorem 1.1 in subsection 4.2). First, we present the implementation of the odd-even transposition on buckets. Then we will deal with I_h. The bucket versions of both networks are shown in Figure 5.1. A basic component of the net-

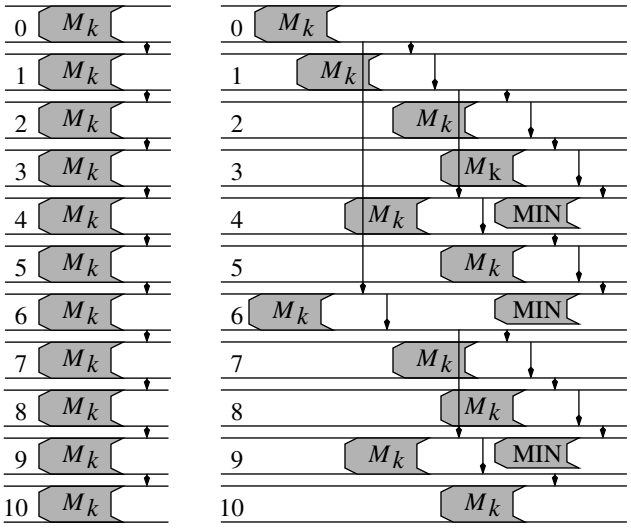


FIG. 5.1. *Examples of $T_{N,k}$ and $J_{N,k}$ networks—the bucket versions of odd-even transposition and the I_2 network.*

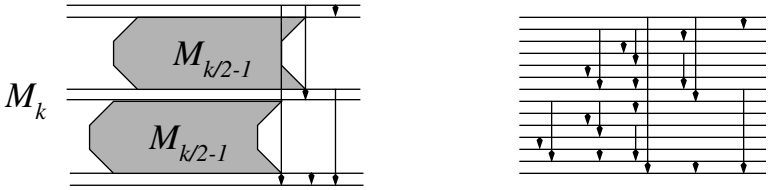


FIG. 5.2. *The recursive structure of M_k and the M_{14} network.*

works inside buckets is a MINMAX-network M_k that is described in Lemma 5.1. The next two lemmas give the correcting properties of our bucket INSERT-networks. In Lemma 5.2, we consider a single copy of the bucket version of I_h. This is an analogue of Lemma 4.3. Finally, Lemma 5.3 summarizes the properties of the whole construction.

LEMMA 5.1. *For each $k \geq 2$ there is an explicit construction of a standard k-input MINMAX-network M_k such that $depth(M_k) \leq 2 \log k$ and $delay(M_k) \leq 4$.*

Proof. The recursive construction of M_k that we give is applicable for $k \geq 6$ (see also Figure 5.2). The construction of the network for the initial values of k is straightforward: $M_2 = \{[0:1]\}$, $M_3 = \{[0:1]\}\{[0:2]\}\{[1:2]\}$, $M_4 = \{[0:3],[1:2]\}\{[0:1],[2:3]\}$, and $M_5 = \{[0:4],[1:2]\}\{[0:1],[3:4]\}\{[0:3],[2:4]\}$. Let $k \geq 6$. It is easier to give the recurrence for the reversed sequence of stages. Therefore, assume that $\mathrm{rev}(A)$ denotes a network A after reversing the sequence of stages of A; that is, $\mathrm{rev}(A) = S_d, S_{d-1}, \ldots, S_1$ if $A = S_1, \ldots, S_d$:

$$\mathrm{rev}(M_k) \stackrel{\mathrm{def}}{=} (\{[0:1],[\lceil k/2\rceil - 1 : k-1]\},\{[0:\lceil k/2\rceil],[k-2:k-1]\},\{[0:k-1]\}) \\ \cup (\mathrm{rev}(M_{\lceil k/2\rceil - 1})^{\Rightarrow 1} \Downarrow 1) \cup (\mathrm{rev}(M_{\lfloor k/2\rfloor - 1})^{\Rightarrow 2} \Downarrow \lceil k/2\rceil).$$

Note that in order to obtain $delay(M_k) \leq 4$, the upper subnetwork $M_{k/2-1}$ is shifted one position to the right compared to the lower one. The reason is that registers of the upper subnetwork are used outside it in the last stage only, while registers of

the lower one are used one stage earlier. Clearly, the network is correctly defined. Moreover, the depth of M_k is bounded by

$$\max(\mathit{depth}(M_{\lceil k/2\rceil-1})+1, \mathit{depth}(M_{\lfloor k/2\rfloor-1})+2),$$

and this is at most $2\log(k/2)+2 = 2\log k$. To see that $\mathit{delay}(M_k)$ is at most 4, notice that the external registers 0 and $k-1$ are used only in the last three stages. □

FACT 5. *For $k > 2$ and $j > 0$ let N_k denote the j-fault subnetwork of M_k. If for some input $\overline{x} \in 1\{0,1\}^{k-2}0$ the output of N_k has the property $(\overline{x}N_k)[0] = 1$ and $(\overline{x}N_k)[k-1] = 0$, then $j \geq 2$.*

Proof. Assume to the contrary that the conditions from Fact 5 hold and that $j = 1$. Then $[0 : k-1]$ is the only faulty comparator in N_k. Consider the initial contents of the register 1, which is different from registers 0 and $k-1$, when $k > 2$. If $\overline{x}[1] = 0$, then the comparator $[0:1]$ must also be faulty. If $\overline{x}[1] = 1$, then this 1 is moved to the register $\lceil k/2\rceil - 1$ (or 2 if $k = 4$), and then also the comparator $[\lceil k/2\rceil - 1 : k-1]$ ($[2:3]$, respectively) must be faulty—a contradiction. □

Informally speaking, Fact 5 says that to destroy both MIN and MAX outputs of M_k we need at least two faulty comparators if $k > 2$. For $k = 2$ we obtain the same effect when we repeat the network $M_2 = \{[0:1]\}$ twice.

To make all our notation simpler, from now on we assume that $k > 2$ and $N = pk$, that is, that the set of registers can be exactly divided into p buckets of k registers each.[4] Thus, we are dealing with the INSERT-network of p inputs. For $m = 0, 1, \ldots, p-1$, let $l_m = mk$ and $u_m = (m+1)k - 1$. Define transportation network $T_{N,k}$ (compare Figure 5.1) to be

$$T_{N,k} \stackrel{\text{def}}{=} \{[u_{m-1} : l_m] : 0 < m < p\}^{\Rightarrow \mathit{depth}(M_k)} \cup \bigcup_{m=0}^{p-1} (M_k \Downarrow l_m).$$

Note that $\mathit{depth}(T_{N,k}) \leq 2\log k + 1$ and $\mathit{delay}(T_{N,k}) \leq 4$. Since $T_{N,k}$ can transport 1^*s between adjacent buckets, to move s such items by the distance of $h+1$ buckets we need $h+1+s$ copies of $T_{N,k}$. If, in addition, j comparators can be faulty, the number of copies increases to $h+1+s+j$. Because, in the case of correction network, $s+j \leq k$, we take $h+k+1$ copies of $T_{N,k}$ with the total depth upper-bounded by $2\log k + 1 + 4(h+k) \leq 4\log N + 4k$.

Now we are going to show how I_h is transformed into the bucket version. Note that the INSERT-network that we need has exactly p inputs; therefore we choose h such that $3 \cdot 2^{h-1} \leq p \leq 3 \cdot 2^h - 1$. If there are superfluous registers in I_h, the last ones should be deleted (cf. the proof of Theorem 1.1).

To describe the transformation, we need the structure of incoming and outgoing comparators of each register in I_h. Recall that comparators involving a register i can only appear at stages $S_{\mathit{fst}(i,I_h)}, S_{\mathit{fst}(i,I_h)+1}, \ldots, S_{\mathit{lst}(i,I_h)}$. By simple induction one can check that the sequence of comparators for a register i satisfies the following properties.

FACT 6. *For each register i, $1 \leq i \leq 3 \cdot 2^h - 1$, in the network $I_h = S_1, S_2, \ldots, S_d$ the following hold:*

1. *if $[i : *] \in S_{\mathit{fst}(i,I_h)}$, then $\mathit{lst}(i, I_h) \leq \mathit{fst}(i, I_h) + 1$ and $[* : i] \notin S_{\mathit{fst}(i,I_h)+1}$;*
2. *if $[* : i] \in S_{\mathit{fst}(i,I_h)}$, then*
   - *$[* : i] \notin S_{\mathit{fst}(i,I_h)+1} \cup S_{\mathit{fst}(i,I_h)+2}$,*

[4] In the general case, we can take the first (upper) bucket smaller.

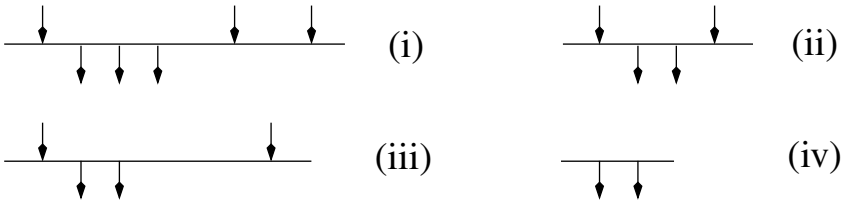


FIG. 5.3. *The four basic cases of a comparator sequence for register i in I_h.*

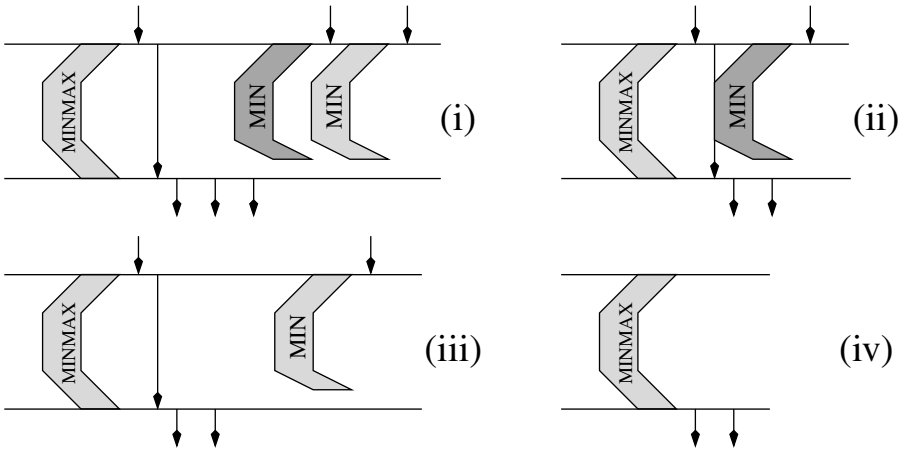


FIG. 5.4. *The structure of buckets in the bucket version of I_h.*

- $i \notin support(S_{fst(i,I_h)+4})$,
- $[i:*] \notin S_{fst(i,I_h)+5} \cup S_{fst(i,I_h)+6} \cup S_{fst(i,I_h)+7}$,
- *if* $[*:i] \in S_{fst(i,I_h)+3}$, *then* $lst(i,I_h) = fst(i,I_h)+3$,
- *if* $[*:i] \in S_{fst(i,I_h)+6}$, *then* $i \notin support(S_{fst(i,I_h)+3} \cup S_{fst(i,I_h)+5} \cup S_{fst(i,I_h)+7})$,
- *if* $[i:*] \in S_{fst(i,I_h)+3}$, *then* $i \notin support(S_{fst(i,I_h)+6})$,
- *if* $[l:i] \in S_{fst(i,I_h)+j}$ *and* $j \geq 1$, *then* $[l:j]$ *is a correcting comparator.*

These properties of the sequence of comparators for a register i are depicted in Figure 5.3. It shows that the sequence is a subsequence of one of the four basic sequences.

Our transformation consists of the following steps:

1. replace each comparator $[i:j]$ by $[u_i:l_j]$, where $u_i = i \cdot k + k - 1$ and $l_j = j \cdot k$;
2. insert an empty stage after each stage of I_h;
3. put the MINMAX-networks M_k, the MIN-networks M'_k (to be described below), and an additional comparator inside buckets, as shown in Figure 5.4.

The M'_k network is obtained from the M_k network by deleting comparators incoming to the last (MAX) register. Let $J_{N,k}$ denote the whole network after the transformation. For example, $J_{11k,k}$ is depicted in Figure 5.1.

To define $J_{N,k}$ formally, let $N = pk$ and $A_p = S_1, S_2, \ldots, S_d$ denote the network I_h, $3 \cdot 2^{h-1} \leq p \leq 3 \cdot 2^h - 1$, after deleting registers $p+1, p+2, \ldots, 3 \cdot 2^h - 1$ and all comparators that use them. Let $B_{N,k} = T_1, T_2, \ldots, T_{2d}$ denote the network after transformations 1 and 2; that means that $T_2 = T_4 = \cdots = T_{2d} = \emptyset$ and $T_{2l-1} = \{[u_i:l_j] : [i:j] \in S_l\}$, $l = 1, 2, \ldots, d$. Moreover, let $C_{k,m}$ be the network put in bucket

m, $0 \le m < p$, that is, $C_{k,m} = M_k \cup \{[0 : k-1]^{\Rightarrow depth(M_k)+1} : [* : m] \in S_{fst(m,A_p)}\} \cup \bigcup\{(M'_k)^{\Rightarrow 2j} : [* : m] \in S_{fst(m,A_p)+j} \wedge j \ge 1\}$. Finally, let

$$J_{N,k} \stackrel{\text{def}}{=} (B_{N,k})^{\Rightarrow depth(M_k)} \cup \bigcup_{m=0}^{p-1} (C_{k,m} \Downarrow l_m)^{\Rightarrow 2fst(m,A_p)-2}.$$

Due to Fact 6, the above definition is correct. One can verify that $depth(J_{N,k}) \le 4\log N + 3$ and $delay(J_{N,k}) \le 18$.

Furthermore, with respect to Fact 5, one faulty comparator inside a MINMAX-network cannot cause both MIN and MAX outputs to be faulty. If such a comparator destroys the MIN output (respectively, MAX output), the effect is as though the next incoming (respectively, outgoing) comparator had been faulty. This effect leads us, informally, to an assumption that only comparators between buckets become faulty and to an analysis of the networks as in the case of INSERT-networks. Formally, we will prove an analogue of Lemma 4.3 for the bucket version of I_h.

Consider an input $\overline{x} \in \{0,1\}^N$ that has l 1^*s among zeroes and $N-t-l$ ones at its end; that is, $\overline{x} = 0^{j_0}10^{j_1}10^{j_2}\dots10^{j_l}1^{N-t-l}$, where $0 \le l \le k$, $j_0, j_1, \dots, j_{l-1} \ge 0$, $j_l \ge 1$, and $\sum_{m=0}^{l} j_m = t$. We say that a comparator $[u_i : l_j]$ between buckets i and j is active with respect to t if $l_j \le t$. Since the structure of active comparators outgoing from a bucket i is the same as in I_h, we can use equality (4.1) to define $c_h(i,t)$, the weight of the bucket i with respect to t. Let $i_1 = \lfloor j_0/k \rfloor, i_2 = \lfloor (j_0 + j_1 + 1)/k \rfloor, \dots, i_l = \lfloor (j_0 + \dots + j_{l-1} + l - 1)/k \rfloor$ denote the indices of buckets that contain 1^*s. Then the weight of $\overline{x}$ is defined to be

$$C_h(\overline{x}, t) \stackrel{\text{def}}{=} \sum_{j=1}^{l} c_h(i_j, t).$$

LEMMA 5.2. *Let $k > 2$, $N = pk$, and h be such that $3 \cdot 2^{h-1} \le p \le 3 \cdot 2^h - 1$. Let $\overline{x} = 0^{j_0}10^{j_1}10^{j_2}\dots10^{j_l}1^{N-t-l}$, where $0 \le l \le k$, $j_0, j_1, \dots, j_{l-1} \ge 0$, $j_l \ge 1$, and $\sum_{m=0}^{l} j_m = t$. Moreover, let $J'_{N,k}$ denote any s-fault subnetwork of $J_{N,k}$ and $\overline{y} = \overline{x}J'_{N,k}$. If the weight $C_h(\overline{x}, t)$ is positive, then*

$$C_h(\overline{y}, t) - C_h(\overline{x}, t) \le s - 1.$$

Proof. Let $J_{N,k} = R_1R_2\dots R_D$ and $J'_{N,k} = R'_1R'_2\dots R'_D$, where $D = depth(J_{N,k})$ and $R'_i \subseteq R_i$, $i = 1, \dots, D$. Let $\overline{x}_i = \overline{x}R'_1R'_2\dots R'_i$, $i = 0, \dots, D$. We say that a comparator $[u_i : l_j] \in R_m$ between buckets i and j is an *exchanging* comparator with respect to $J'_{N,k}$ and $\overline{x}$ if it really exchanges 0 with 1, that is, if $[u_i : l_j] \in R'_m$ and $\overline{x}_{m-1}[u_i] = 1$ and $\overline{x}_{m-1}[l_j] = 0$ (hence $\overline{x}_m[u_i] = 0$ and $\overline{x}_m[l_j] = 1$). Only exchanging comparators move ones among buckets and, therefore, can change the weight of the sequence $\overline{x}_i$. Let $E(J'_{N,k}, \overline{x})$ denote the set of all exchanging comparators with respect to $J'_{N,k}$ and $\overline{x}$. One can easily observe that

$$C_h(\overline{y}, t) - C_h(\overline{x}, t) = \sum_{[u_i:l_j] \in E(J'_{N,k}, \overline{x})} c_h(j, t) - c_h(i, t).$$

For each i, there is at most one exchanging comparator $[u_i : *]$ outgoing from a bucket i, since all outgoing comparators are applied one after another (cf. Figure 5.4). To prove the lemma, we would like to assign to each bucket i a set of faulty comparators F_i such that $F_i = \emptyset$ if there are no exchanging comparators outgoing from i, or

else it contains enough elements to get $c_h(j,t) - c_h(i,t) \leq |F_i|$, where $[u_i : l_j]$ is the exchanging comparator. In addition, all these sets should be pairwise disjoint.

Let f_i be the number of active comparators outgoing from a bucket i that precede an exchanging comparator $[u_i : l_j]$. By the definition of c_h, $c_h(i,t) \geq c_h(j,t) - f_i$. Therefore, we have only to choose f_i faulty comparators for F_i. Since the preceding comparators do not move the 1 located in the ith MAX-register down, they have to be faulty or blocked by another 1 in their destination registers. The faulty comparators are inserted directly into F_i. In case of a blocked comparator, there is a 1 in the MIN-register of its destination bucket. The bucket cannot be full, because of our definition of active comparators. That means that there must be a faulty comparator in the subnetworks M_k or M'_k which precede the comparator in question in the destination bucket. We make it also a member of F_i. Sets F_i, $i = 0, 1, \ldots, p-1$, are pairwise disjoint, because each faulty active comparator is inserted into at most one set F_i, and the faulty comparators responsible for blocking 1's are selected from different subnetworks.

Summing up the inequalities, we would get $C_h(\overline{y},t) - C_h(\overline{x},t) \leq \sum_{i=0}^{p-1} f_i \leq s$. In order to get the stronger upper bound $s-1$, we will show that at least one of the following situations always must occur:

(i) there is an exchanging comparator that is also a correcting comparator;

(ii) there is an exchanging comparator that moves a 1 from a bucket with some positive weight to a bucket with zero weight;

(iii) there exists a faulty comparator outside $\bigcup_{i=0}^{p-1} F_i$.

In cases (i) and (ii), for at least one exchanging comparator $[u_i : l_j]$ we have a stronger upper bound $c_h(j,t) - c_h(i,t) \leq -1$. The result follows. In case (iii), obviously, $\sum_{i=0}^{p-1} f_i \leq s-1$. To finish the proof, we assume that (i) and (ii) do not hold, and we prove (iii).

Consider a bucket with the highest index i_{max} such that $c_h(i_{max},t) > 0$ and such that bucket contains at least one 1 at the moment when the sequence $\overline{y}$ is in the registers. Such a bucket must exist, because (ii) does not hold and there are buckets with positive weights at the moment when $\overline{x}$ is in registers. Since $c_h(i_{max},t) > 0$, there are active comparators outgoing from i_{max}, but none of them is an exchanging comparator, due to our assumption. This can only be caused by one of the following situations:

1. The bucket i_{max} contained 1's at the beginning, but no 1 was moved to the MAX-register due to faulty comparators in the subnetwork M_k. Due to Fact 5, we can select a faulty comparator which is not in $\bigcup_{i=0}^{p-1} F_i$.
2. The bucket i_{max} contained a 1 at the beginning that was moved to the MAX-register, but all active comparators outgoing from i_{max} were faulty or blocked. We can select at least one faulty comparator as in the case of F_i. Recall that $F_{i_{max}}$ is empty in this case.
3. The bucket i_{max} contained no 1's at the beginning. The 1 which is in the bucket at the end could only be moved there by the first incoming comparator (the second and the third one, if they exist, are correcting and could not be exchanging due to the assumption that (i) does not hold). The 1 is blocked in the bucket by the next faulty comparator $[l_{i_{max}} : u_{i_{max}}]$ or by faulty/blocked outgoing comparators, as in the previous case. □

Repeating the further arguments from section 4, we get that if an $\overline{x}$ with l 1*s is given as an input to the bucket INSERT-network and $C_h(\overline{x},t) \leq l$ holds, then k copies of $J_{N,k}$ are enough to obtain an output $\overline{y}$ with $C_h(\overline{y},t) = 0$ and to have a network resistant to $k-l$ faults. Since buckets of weight 0 are at the distance of at most $h+1$

buckets from the zero/one border, the previously described network $T_{n,k}^{(h+k+1)}$ will gather all 1^* in one "border" bucket. To end up with the sorted sequence, we must only sort the bucket. Thus, the bucket fault-tolerant INSERT-network is defined to be

$$L_{N,k} \stackrel{\text{def}}{=} J_{N,k}^{(k)} \cup (T_{n,k}^{(h+k+1)})^{\Rightarrow depth(J_{N,k}^{(k)})}.$$

The total depth of the network is bounded by $4\log N + 3 + 18(k-1) + 4\log N + 4k = O(\log N + k)$. An output of the network will be sorted by the third part of our CORRECT 1^*s network, namely, by $3k$ stages of odd-even transposition. Formally, we have the following lemma.

LEMMA 5.3. *Let $k > 2$, $N = pk$, and h be such that $3 \cdot 2^{h-1} \le p \le 3 \cdot 2^h - 1$. Let $\overline{x} = 0^{j_0}10^{j_1}10^{j_2}\ldots10^{j_l}1^{N-t-l}$, where $0 \le l \le k$, $j_0, j_1, \ldots, j_{l-1} \ge 0$, $j_l \ge 1$, and $\sum_{m=0}^{l} j_m = t$. Moreover, let $0 \le s \le k-l$, $L'_{N,k}$ denote any s-fault subnetwork of $L_{N,k}$, and $\overline{y} = \overline{x}L'_{N,k}$. If the weight $C_h(\overline{x}, t)$ is not greater than l, then the output $\overline{y}$ is sorted except for the contents of bucket $t' = \lfloor t/k \rfloor$; that is,*

$$\Big[(y[0], \ldots, y[kt'-1]) = 0^{kt'} \wedge (y[k(t'+1)], \ldots, y[N-1]) = 1^{N-k(t'+1)}\Big].$$

Proof. Let s_i, $0 < i \le k$, denote the number of faulty comparators in the ith copy of $J_{N,k}$, and let $s' = s - \sum_{i=1}^{k} s_i$. Moreover, let $\overline{x}_i$ be the contents of registers after the execution of the ith (faulty) copy of $J_{N,k}$, and let $\overline{x}_0 = \overline{x}$. First, we would like to prove that $C_h(\overline{x}_k, t)$ is zero. To this end, assume to the contrary that $C_h(\overline{x}_k, t) > 0$. Due to the definition, all weights $C_h(\overline{x}_i, t)$, $0 \le i \le k$, must be positive. By Lemma 5.2, for each i, $0 \le i \le k$, we have

$$C_h(\overline{x}_i, t) - C_h(\overline{x}_{i-1}, t) \le s_i - 1.$$

Summing up the inequalities, we get $C_h(\overline{x}_k, t) - C_h(\overline{x}_0, t) \le (\sum_{i=1}^{k} s_i) - k$. Since $C_h(\overline{x}_0, t) \le l$ and $s + l \le k$, we finally get $C_h(\overline{x}_k, t) \le 0$, which is a contradiction.

Let us define the *bucket distance* between registers i and j to be $\lfloor j/k \rfloor - \lfloor i/k \rfloor$. The *bucket length* of a comparator $[i : j]$ is the bucket distance between i and j. Obviously, the bucket length of $[u_i : l_j]$ is equal to the length of $[i : j]$. Due to Fact 2, the bucket distance between a register i such that $i < t$ and $\overline{x}_k[i] = 1$ and the register t is at most h. Since $\lfloor t/k \rfloor + 1$ is the index of the last bucket that can contain zeroes in $\overline{x}_k$, the goal of an s'-fault subnetwork of $T_{N,k}^{(h+k+1)}$ is to transport all (at most l) 1^*s by the bucket distance of at most $h+1$.

In the s'-fault subnetwork of $T_{N,k}^{(h+k+1)}$ there are at least $h + k + 1 - s' \ge h + l + 1$ fault-free copies of $T_{N,k}$. The standard analysis of odd-even transposition among consecutive buckets (cf. [6, pp. 139–144]) shows that this number of transposition levels is enough to fill up bucket $t' + 1$ and gather all other 1^*s in bucket t'. □

In the next section, we describe a network that moves 1^*s to buckets whose weights are bounded by 1 each.

6. The movement network. The aim of the movement network is to place all 1^*s in buckets with weights at most one. The idea of the movement is the same as in the construction of Schimmler and Starke [12]. It can be summarized as follows:

(i) comparators between buckets create a forest of binary trees, where the goal of each tree is to place the maximum of the numbers in its nodes in the root;

(ii) the maximum in a binary tree is sought by applying in each node the following simple procedure: compare-exchange numbers with its children and then with its parent.

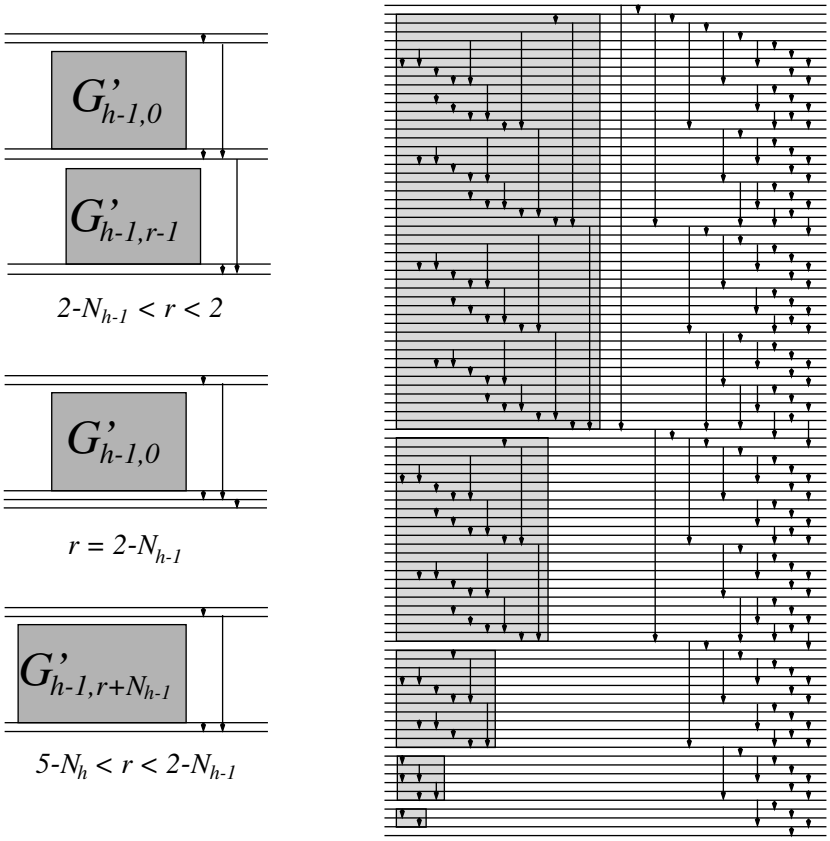


FIG. 6.1. *The recursive structure of $G_{h,r}$ and the F_5 network in front of I_5.*

In our case, the structure of binary trees should reflect the structure of I_h defined in section 4. For example, registers in I_h should be divided into the following groups: the register number 0 has always the weight at most 1; registers $1, 2, \ldots, N_{h-1}$ can have bigger weights, and therefore we should construct a binary tree on them with the root in $N_{h-1} + 1$, where the weight is again at most 1; and so on. In general, our forest is defined on $N_h = 3 \cdot 2^h - 1$ registers as

$$F_h \stackrel{\text{def}}{=} \bigcup_{i=1}^{h} G_{h-i,+1} \Downarrow (N_h - N_{h-i+1} + 1).$$

Before we formally define the subnetworks of F_h, let us look for a moment at the sample network F_5 in Figure 6.1. It consists of 95 registers numbered $0, 1, \ldots, 94$. Assume that we put an input sequence $0^i 10^{45-i} 1^{49}$, $0 \leq i < 45$, into them. To see what F_5 is doing, let us try first to answer the following question: If F_5 were not in front of I_5, what would be the values of i for which the input would be sorted by one fault-free copy of I_5?

For the inputs under consideration, active comparators of I_5 are all above the border between registers 45 and 46. Therefore, we have only one sequence of consecutive active comparators that ends at register 45, namely, the sequence $[0 : 1]$, $[1 : 25]$, $[25 : 37]$, $[37 : 43]$, $[43 : 44]$, $[44 : 45]$. Thus the answer is that i should be in the

set $\{0,1,25,37,43,44,45\}$. If i is not in this set, we need F_5 in front of I_5: the aim of F_5 is to move 1 from any input register i, $0 \le i < 45$, to the output register in the set. For example, if $1 \le i \le 25$, then the output is in register 25; if $26 \le i \le 37$, then the output is in register 37; and so on. Note that for each i in the set we have $c_5(i,45) \le 1$, where c_h is the weight function defined in section 4.

$G_{h,r}$ is a network corresponding to a binary tree and is defined on $N_h + r$ registers, where an integer $r \le 1$ can be negative; it is a MAX-network. Since it should reflect the structure of I_h and, moreover, should have a small delay, the sons of the root $N_h + r - 1$ are $N_h + r - 2$ and $N_{h-1} + 1$. In addition, 0 should be a son of 1. Formally, we will define $G_{h,r}$ with the help of another recursively defined MAX-network $G'_{h,r}$, which does not change the contents of register 0 and finds the maximum value in all other registers. Let $G'_{0,+1} \stackrel{\text{def}}{=} \{[1:2]\}$ and $G'_{0,r} \stackrel{\text{def}}{=} \emptyset$ for $r \le 0$. Then

$$G_{h,r} \stackrel{\text{def}}{=} \begin{cases} G'_{h,r} \cup \{[0:1]\}^{\Rightarrow depth(G'_{h,r})-3}, & 1 - N_{h-1} < r \le 1, \\ G'_{h,r} \cup \{[0:1]\}^{\Rightarrow depth(G'_{h,r})-2}, & 3 - N_h < r \le 1 - N_{h-1}, \\ G'_{h,r} \cup \{[0:1]\}, & r = 3 - N_h, 2 - N_h, \\ \emptyset, & r < 2 - N_h. \end{cases}$$

As in the case of MINMAX-network M_k, it would be easier to define the sequence of stages of $G'_{h,r}$ in reverse order. Let $lst_h = N_h + r - 1$, $mid_h = N_{h-1} + 1$, and

$$B_h^1 \stackrel{\text{def}}{=} \{[mid_h : lst_h]\}\{[1 : mid_h], [lst_h - 1 : lst_h]\}\{[mid_h - 1 : mid_h]\},$$
$$B_h^2 \stackrel{\text{def}}{=} \{[lst_h - 1 : lst_h]\}\{[1 : lst_h - 1]\}\{[lst_h - 2 : lst_h - 1]\},$$
$$B_h^3 \stackrel{\text{def}}{=} \{[1 : lst_h]\}\{[lst_h - 1 : lst_h]\},$$

$$\operatorname{rev}(G'_{h,r}) \stackrel{\text{def}}{=} \begin{cases} B_h^1 \cup \operatorname{rev}(G'_{h-1,r-1})^{\Rightarrow 2} \Downarrow N_{h-1} \cup \operatorname{rev}(G'_{h-1,0})^{\Rightarrow 3} \Downarrow 1, & \\ & 2 - N_{h-1} < r \le 1, \\ B_h^2 \cup \operatorname{rev}(G'_{h-1,0})^{\Rightarrow 3} \Downarrow 1, & r = 2 - N_{h-1}, \\ B_h^3 \cup \operatorname{rev}(G'_{h-1,N_{h-1}+r})^{\Rightarrow 2} \Downarrow 1, & 3 - N_h < r \le 1 - N_{h-1}, \\ \{[1:2]\}, & r = 3 - N_h, \\ \emptyset, & r < 3 - N_h. \end{cases}$$

Both networks $G_{h,r}$ and $G'_{h,r}$ have depth not greater than $3h$ since we add at most three new stages at each level of the recursive definition. The delay of the networks is at most 3, because each register has at most two incoming and one outgoing comparators at three consecutive stages. Clearly, $G_{h,r}$ is a MAX-network, and $G'_{h,r}$ is a MAX-network of registers $1, \ldots, N_h + r - 1$. The following lemma states that, whenever F_h is applied to an input with one 1^* inside t zeroes, the 1^* is output in register j such that $c_h(j,t) \le 1$.

LEMMA 6.1. *Let $h \ge 0$, $2 - 3 \cdot 2^h \le r \le 1$, and let X denote one of the networks F_h, $G_{h,+1}$, or $G'_{h,r}$. Moreover, let $t \ge 1$ be smaller than the number of registers in X. Then for each input $\overline{x} = 0^i 10^{t-i} 1^*$, $0 \le i < t$, the output $\overline{y} = \overline{x}X$ is of the form $0^j 10^{t-j} 1^*$, $0 \le j \le t$, and $c_h(j,t) \le 1$.*

Proof. The proof is by induction on h. If $h = 0$, there is at most one active comparator in X. Therefore, for each register j of X and any admissible t, we have $c_0(j,t) \le 1$ (cf. Definition 4.1).

Assume now that $h > 0$ and that the lemma holds for $h - 1$. Observe first that $F_h = G_{h-1,+1} \cup F_{h-1} \Downarrow N_{h-1}$. If $t \le N_{h-1}$, then active comparators in I_h are the same

as in I_{h-1}. Thus $c_h(j,t) = c_{h-1}(j,t)$ for any $1 \le j \le t$, and we can use an inductive hypothesis about $G_{h-1,+1}$. Otherwise, $t \ge N_{h-1}+1$, and the network $G_{h-1,+1}$ gets at most one 1 in its registers. Since $G_{h-1,+1}$ is a MAX-network, it moves the 1 to the register $N'_h = N_{h-1}+1$. Thus we should prove that in this case $c_h(N'_h,t) \le 1$. To this end we need the recursive structure of c_h observed in the proof of Lemma 4.5—$c_h(j,t) = c_{h-1}(j-N'_h, t-N'_h)$ for $N'_h \le j \le t < N_h$—and the first statement from the lemma: $c_h(0,t) \le 1$. Substituting $j = N'_h$, we get the result. If there is no 1 in the first N'_h+1 registers, the problem is reduced to the network F_{h-1} and the lemma follows from the inductive assumption.

Since quite similar arguments also work for networks $G_{h,+1}$ and $G'_{h,r}$, we leave the rest of the proof to the interested reader. □

Having the network F_h, we can create its bucket version by applying a transformation similar to that of section 5. As in the definitions there, assume that $N = pk$, where p is an integer, and choose h such that $3 \cdot 2^{h-1} \le p \le 3 \cdot 2^h - 1$. Since we now need a forest F_h with p registers only, we delete the superfluous registers $p, p+1, \ldots, 3 \cdot 2^h - 1$ and the comparators pointing to them. After that the following transformations are made to the network:

1. replace each comparator $[i:j]$ by $[u_i : l_j]$, where $u_i = i \cdot k + k - 1$ and $l_j = j \cdot k$;
2. insert an empty stage after each stage of F_h;
3. put the MINMAX-networks M_k in front of the first comparator in each bucket j and an additional comparator $[l_j : u_j]$ between incoming and outgoing comparators of the bucket. (Note that in each bucket there is at most one outgoing comparator and at most two incoming ones that precede it.)

Let $K_{N,k}$ be the network after the transformations. Its delay is still a constant, and we can use copies of $K_{N,k}$ to move all 1^*s to buckets of weight at most 1. Moreover, we have a network resistant to faulty comparators. Evidently, it is enough to take k copies, since the sum of the number of faulty comparators in $K^{(k)}_{N,k}$ and the number of 1^* in an input is not greater than k.

LEMMA 6.2. *Let $k \ge 2$, $N = pk$, and h be such that $3 \cdot 2^{h-1} \le p \le 3 \cdot 2^h - 1$. Let $\overline{x} = 0^{j_0}10^{j_1}10^{j_2}\ldots10^{j_l}1^{N-t-l}$, where $0 \le l \le k$, $j_0, j_1, \ldots, j_{l-1} \ge 0$, $j_l \ge 1$, and $\sum_{m=0}^{l} j_m = t$. Moreover, let $K'_{N,k}$ denote any s-fault subnetwork of $K^{(k)}_{N,k}$, where $s \le k - l$. Then $C_h(\overline{x}K'_{N,k}, t) \le l$.*

Proof. One can observe that there are at least l fault-free copies of $K_{N,k}$ in $K'_{N,k}$. Each fault-free copy takes a 1^* from the last nonempty bucket with weight greater than 1 and moves it to a bucket with weight less than or equal to 1, due to Lemma 6.1. From its destination bucket there is no active outgoing comparator. Therefore, faulty copies cannot move such a 1^* further. After application of all fault-free copies, all l 1^*s are in buckets with weight upper-bounded by 1. The result follows. □

Based on Lemma 6.2, we can choose $K^{(k)}_{N,k}$ as our bucket movement network. Since $delay(K_{N,k}) \le 9$ and $depth(K_{N,k}) = O(\log N)$, the total depth of the bucket movement network is $O(\log n + k)$.

We finish this section by proving Lemma 3.1.

Proof of Lemma 3.1. Our N-input CORRECT^1_k-network is a concatenation of three networks: $K_{N,k}$ described above, $L_{N,k}$ presented in the previous section, and the standard odd-even transposition network $O_{N,k}$ of depth $3k$ [6, pp. 139–144]. The required correcting and fault-tolerance properties of the first two parts follow from Lemmas 6.2 and 5.3. The aim of $O_{N,k}$ is to sort the remaining unsorted area of size k. In the fault-free model, k stages of odd-even transposition suffice to sort such an area (see [6]). In the k-fault model, there could be up to k faulty stages in

$O_{N,k}$. Nevertheless, there still remain k alternating fault-free stages of odd-even and even-odd comparators that will sort the area. □

7. Concluding remarks. We have presented a construction of N-input k-fault-tolerant sorting networks with asymptotically optimal depth $O(\log N + k)$. The main part of the construction is the fault-tolerant correcting network, which consists of two subnetworks itself: $\mathrm{CORRECT}_k^1$ and $\mathrm{CORRECT}_k^0$. Each subnetwork performs three tasks in sequence: first, it moves the inserted items into selected buckets, then it inserts them into the destination bucket, and finally, it sorts all buckets. The part performing each task has been made fault-tolerant by pipelining a certain number of copies of a constant-delay network. One can observe that instead of pipelining we can get the same result by using in cycle a network of constant depth with output recirculated back as input. Such networks are called *constant-periodic*. Thus, our correcting networks can be implemented as a sequence of constant-periodic networks. An interesting question is whether the whole network can be made constant-periodic without significant increase in correction time. For the fault-free case, efficient constant-periodic sorting networks were presented in [5].

Acknowledgments. The author thanks Yuan Ma for his helpful discussion about the first version of the construction. Many thanks also to anonymous referees and Mirosław Kutyłowski for their comments on the paper.

REFERENCES

[1] M. Ajtai, J. Komlos, and E. Szemeredi, *An $O(n \log n)$ sorting network*, in Proceedings of the 15th Annual ACM Symposium on Theory of Computing, Boston, MA, 1983, ACM, New York, pp. 1–9.

[2] S. Assaf and E. Upfal, *Fault-tolerant sorting networks*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, IEEE Computer Society Press, Los Alamitos, CA, pp. 275–284.

[3] K. E. Batcher, *Sorting networks and their applications*, in Proceedings of the American Federation of Information Processing Societies Spring Joint Computer Conference (AFIPS 1968 SJCC), Vol. 32, AFIPS Press, Montvale, NJ, pp. 307–314.

[4] D. E. Knuth, *The Art of Computer Programming*, 2nd ed., Vol. 3, Addison–Wesley, Reading, MA, 1975.

[5] M. Kutyłowski, K. Loryś, B. Oesterdiekhoff, and R. Wanka, *Fast and feasible periodic sorting networks of constant depth*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, Santa Fe, NM, 1994, IEEE Computer Society Press, Los Alamitos, CA, pp. 369–380.

[6] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*, Morgan-Kaufmann, San Mateo, CA, 1992.

[7] F. T. Leighton, Y. Ma, and C. G. Plaxton, *Breaking the $\Theta(n \log^2 n)$ barrier for sorting with faults*, J. Comput. System Sci., 54 (1997), pp. 265–304.

[8] T. Leighton and Y. Ma, *Tight bounds on the size of fault-tolerant merging and sorting networks with destructive faults*, in Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures, Velen, Germany, 1993, ACM, New York, pp. 30–41.

[9] Y. Ma, *Fault-Tolerant Sorting Network*, Ph.D. thesis, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, 1994.

[10] Y. Ma, *An $O(n \log n)$-size fault-tolerant sorting network*, in Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, Philadelphia, PA, 1996, ACM, New York, pp. 266–275.

[11] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[12] M. Schimmler and C. Starke, *A correction network for N-sorters*, SIAM J. Comput., 18 (1989), pp. 1179–1187.

[13] A. C. Yao and F. F. Yao, *On fault-tolerant networks for sorting*, SIAM J. Comput., 14 (1985), pp. 120–128.

SIAM J. COMPUT.
Vol. 33, No. 6, pp. 1513–1531

ARITHMETIC CIRCUITS AND POLYNOMIAL REPLACEMENT SYSTEMS*

PIERRE MCKENZIE†, HERIBERT VOLLMER‡, AND KLAUS W. WAGNER§

Abstract. This paper addresses the problems of counting proof-trees (as introduced by Venkateswaran and Tompa) and counting proof-circuits, a related but seemingly more natural question. These problems lead to a common generalization of straight-line programs which we call polynomial replacement systems PRSs. We contribute a classification of these systems and we investigate their complexity. Diverse problems falling within the scope of this study include, for example, counting proof-circuits and evaluating $\{\cup, +\}$-circuits over the natural numbers. A number of complexity results are obtained, including a proof that counting proof-circuits is #P-complete.

Key words. arithmetic circuit, counting classes, computational complexity

AMS subject classifications. 68Q05, 68Q17, 11C08

DOI. 10.1137/S009753970139207X

1. Introduction.

1.1. Motivation. When + and × replace ∨ and ∧ in Figure 1, the gate g_1 on input $x_1 = x_2 = 1$ evaluates to 9. Equivalently, the tree-like Boolean circuit T obtained from the circuit in Figure 1 has nine *proof-trees* [24], i.e., nine different minimal subcircuits witnessing that T outputs 1 (gates replicated to form T are independent). This relationship between proof-tree counting and monotone arithmetic circuits was used by Venkateswaran [23] to characterize nondeterministic time classes, including #P [22], and by Vinay [25] to characterize the counting version of LOGCFL [20]. The same relationship triggered the investigation of $\#\mathrm{NC}^1$ by Caussinus et al. [6] and that of $\#\mathrm{AC}^0$ by Agrawal, Allender, and Datta [1]. See [4, 14, 2] for further results and for motivation to study such "small" arithmetic classes.

A recent goal has been to capture small arithmetic classes by counting objects other than proof-trees, notably paths in graphs. Allender et al. [3] succeeded in identifying appropriate graphs for $\#\mathrm{AC}^0$. Given the growing importance of counting classes, our motivation for the present work was the desire to avoid unwinding circuits into trees before counting their "proofs." Define a proof-*circuit* to be a minimal subcircuit witnessing that a circuit outputs 1. More precisely, for a Boolean circuit C and an input x, a proof-circuit is an edge-induced connected subcircuit of C which evaluates to 1 on x. This subcircuit must contain the output-gate of C, as well as exactly one C-edge into each ∨-gate and all C-edges into each ∧-gate. The reader should convince herself that the circuit depicted above, which had nine proof-trees on input $x_1 = x_2 = 1$, has only seven proof-circuits on that input.

*Received by the editors July 10, 2001; accepted for publication (in revised form) April 6, 2004; published electronically September 2, 2004.
http://www.siam.org/journals/sicomp/33-6/39207.html

†Informatique et Recherche Opérationnelle, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal QB, H3C 3J7 Canada (mckenzie@iro.umontreal.ca). This author's research was performed in part while he was on leave at the Universität Tübingen, and was supported by the (German) DFG, the (Canadian) NSERC, and the (Québec) FQRNT.

‡Theoretische Informatik, Universität Hannover, Appelstraße 4, 30167 Hannover, Germany (vollmer@thi.uni-hannover.de).

§Theoretische Informatik, Universität Würzburg, Am Exerzierplatz 3, 97072 Würzburg, Germany (wagner@informatik.uni-wuerzburg.de). This author's research was supported by DFG grant Wa 847/1-2.

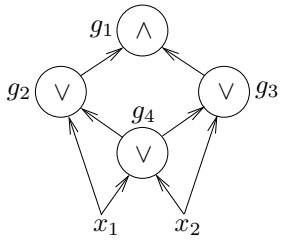


FIG. 1.

What counting classes arise from counting proof-circuits instead of trees? This question held in stock two surprises, the first of which is the following algorithm.

1. replace $\vee$ by $+$ and $\wedge$ by $\times$ in a negation-free Boolean circuit C,
2. view C as a straight-line program prescribing in the usual way a formal polynomial in the input variables $x_1, \ldots, x_n$,
3. compute the polynomial top-down, with an important proviso: at each step, knock any nontrivial exponent down to 1 in the intermediate sum-of-monomials representation.

We get the number of proof-circuits of C on an input x by evaluating the final polynomial at x! For example, the circuit depicted above had seven proof-circuits on input $x_1 = x_2 = 1$ because

$$g_1 \to g_2 g_3 \tag{1.1}$$

$$\to (x_1 + g_4) g_3 \tag{1.2}$$

$$\to (x_1 + g_4)(g_4 + x_2) \quad \to \quad x_1 g_4 + x_1 x_2 + g_4 + g_4 x_2 \tag{1.3}$$

$$\to x_1(x_1 + x_2) + x_1 x_2 + (x_1 + x_2) + (x_1 + x_2) x_2, \tag{1.4}$$

where g_4^2 became g_4 in the middle of step (1.3).

One's intuition might be that such a simple strategy could be massaged into an arithmetic circuit or at least into a sublinear parallel algorithm [21]. Our second surprise was that counting proof-circuits, even for depth-4 semiunbounded circuits, is #P-complete. Hence, not only is our strategy hard to parallelize, but it likely genuinely requires exponential time!

Our three-step algorithm above thus counts proof-*trees* in the absence of the idempotent rules $y^2 \to y$, and it counts proof-*circuits* in their presence. Moreover, whereas an arithmetic circuit computing the number of proof-*trees* of a circuit is readily available, producing such a circuit to compute proof-*circuits* seems intractable. What is special about the idempotent rules? What would the effect of multivariate rules be? Which nontrivial rules would nonetheless permit expressing the counting in the form of an arithmetic circuit? What is a general framework in which complexity questions such as these can be investigated?

1.2. Results. We view our results as forming three main contributions.

Our first contribution is to define and classify *polynomial replacement systems* (PRSs). PRSs provide the answer to the framework question. A PRS in its full generality is a start polynomial $q \in \mathbb{N}[x_1, \ldots, x_m]$ together with a set of replacement rules. A replacement rule is a pair of polynomials (p_1, p_2). Informally, (p_1, p_2) is applicable to a polynomial q if q can be written in a form in which p_1 appears. Applying (p_1, p_2) to q then consists of replacing p_1 by p_2 (see section 3 for formal definitions).

A PRS generally defines a *set* of polynomials, since the choice and sequencing of the rules, and the way in which the rules are applied, may generate different polynomials. Computational problems of interest include computing the polynomials themselves (POLY), evaluating the polynomials at specific points (EVAL), and testing membership in their ranges (RANGE). We identify four natural families of PRSs: *simple* if the rules replace only variables, *deterministic* if no two rules have the same left-hand side, *acyclic* if no nontrivial infinite sequence of rules is applicable, and *idempotent* if the rules (y^2, y) are present.

For general PRSs, we obtain canonical forms, we discuss representation, and we outline broad complexity issues. Our detailed complexity analysis involves *simple* PRSs. For instance, we exhibit simple and deterministic PRSs for which RANGE is NP-complete. When the simple and deterministic PRS is given as part of the input, POLY is P-hard and in coRP, while RANGE is NP-complete and EVAL is P-complete.

Our second contribution concerns the specific case of proof-trees and proof-circuits. We prove that, for every Boolean circuit C and input x, there is an easily computable, idempotent, simple, deterministic, and acyclic PRS S having the property that the number of proof-trees (resp., proof-circuits) of C on x is the maximum (resp., minimum) value of the EVAL problem for S on x, and vice versa (see Lemma 5.13). This offers one viewpoint on the reason why our algorithm from subsection 1.1 counts proof-circuits correctly. We also prove that computing the minimum of the EVAL problem for idempotent, simple, deterministic, and acyclic PRSs is #P-complete, or, equivalently, that counting proof-circuits is #P-complete under Turing reductions (but not under many-one reductions unless P = NP). This provides a new characterization of #P which is to be contrasted with Venkateswaran's (polydegree, polydepth) characterization [23] and with the retarded polynomials characterization of Babai and Fortnow [5]. We also prove that detecting whether a circuit has more proof-trees than proof-circuits is NP-complete.

Our third contribution concerns the specific case of a simple and acyclic PRS. We prove that the EVAL problem for such a PRS is the evaluation problem for $\{\cup, +, \times\}$-circuits. These circuits have been considered previously (under the name hierarchical descriptions) in [28, 29]. They are obtained by generalizing, from trees to general circuits, the $\{\cup, +, \times\}$-expressions (a.k.a. integer expressions), whose evaluation problem was shown to be NP-complete by Stockmeyer and Meyer [19]. From a PSPACE upper bound given in [28] we conclude that evaluation of a simple acyclic PRS has a polynomial space algorithm, and from a PSPACE-hardness result given in [30] we then conclude that the problem is PSPACE-complete.

1.3. Paper organization. Section 2 defines proof-trees and circuits, and proves complexity results about them, in a self-contained way independent from PRSs. The formal definition of a PRS, as well as their canonical forms, are found in section 3. Section 4 briefly discusses the representation of polynomials and the complexity of equivalence testing. Section 5 describes the four natural families of PRSs and relates these to straight-line programs, $\{\cup, +, \times\}$-circuits, and the proof-trees vs. proof-circuits problem. Section 6 investigates the complexity theoretic properties of simple PRSs in depth.

2. Counting circuits vs. counting trees. We assume that the reader is minimally familiar with the complexity classes TC^0, P, ZPP, RP, NP, PSPACE, EXPTIME, FP, and #P; see, for example, [9, 17, 27].

2.1. Definition of problems. By a circuit C, in this paper, we will mean a circuit over the basis $\{\wedge, \vee\}$ in the usual sense, with $2n$ inputs labeled $x_1, x_2, \ldots, x_n, \neg x_1, \neg x_2, \ldots, \neg x_n$.

Fix an input x to C. Unwind C into a tree C' by (repeatedly) duplicating gates with fan-out greater than 1. Define an *accepting subtree* as a subgraph H of C' whose gates evaluate to 1 and which additionally fulfills the following properties: Subtree H must contain the output-gate of C. For every $\wedge$-gate v in H, all the input wires of v must be in H, and for every $\vee$-gate v in H, exactly one input wire of v must be in H. Only wires and nodes obtained in this way belong to H. By $\#C(x)$ we denote the number of accepting subtrees of C.

Define an *accepting subcircuit* as a subcircuit H of C with the same properties as above (i.e., the only difference is that now we do *not* start by unwinding C into a tree). Given an input x, let $\#_c C(x)$ denote the number of accepting subcircuits of C on x.

Since an accepting subtree or subcircuit of a circuit C is some form of proof that C evaluates to 1 on the given input, we will also refer to these as *proof-trees* and *proof-circuits*, respectively. We will consider the following problems:

Problem: PT
Input: circuit C over $\{\wedge, \vee\}$, an input $x \in \{0,1\}^*$, a number k in unary
Output: $\#C(x) \bmod 2^k$

Problem: PC
Input: circuit C over $\{\wedge, \vee\}$, an input $x \in \{0,1\}^*$
Output: $\#_c C(x)$

Observe that if we unwind a circuit into a tree, there may be an exponential blowup in size, which has the consequence that the number of accepting subtrees may be doubly exponential in the size of the original circuit. This is not possible for the problem PC; the values of this function can be at most exponential in the input length. In order to achieve a fair comparison of the complexity of the problems, we therefore count proof-trees only modulo an exponential number.

2.2. Some initial reductions.

LEMMA 2.1. PT *is complete for* FP *under* $\leq_m^{\log}$.

Proof. By the well-known connection between counting accepting subtrees and the evaluation of arithmetic circuits [26, 23, 11], PT $\in$ FP since it is sufficient to evaluate such a circuit modulo an exponential number.

Let now $f \in$ FP and $x \in \{0,1\}^*$. Let f for inputs of length $|x|$ be computed by the polynomial size circuit C. Let $g_1, \ldots, g_m$ be the output-gates of C, and let $D_1, \ldots, D_m$ be those subcircuits of C whose output-gates are the gates $g_1, \ldots, g_m$. Say that C is unambiguous if for every circuit D_i there is at most one accepting subtree. We may suppose without loss of generality that C is unambiguous [13]. Now let C_i be the trivial circuit with 2^i accepting subtrees. Then $f(x)$ is equal to the number of accepting subtrees of the circuit $\bigvee_{i=1}^{m}(D_i \wedge C_i)$. □

LEMMA 2.2.

1. *The following problem is* NP*-complete under* $\leq_m^{\log}$: *Given a circuit* C*, is there an input* x *such that* $\#C(x) \neq \#_c C(x)$*?*

2. *The following problem is* P*-complete under* $\leq_m^{\log}$: *Given a circuit* C *and an input* $x \in \{0,1\}^*$*, is* $\#C(x) \neq \#_c C(x)$*?*

Proof. 1. Containment in NP is obtained by the following algorithm: On input C, guess an input x and evaluate every gate in C. Check if there is an $\vee$-gate g such

that both its inputs evaluate to 1, and there are at least 2 different paths from g to the output of C which are simultaneously contained in some accepting subtree. This latter problem is essentially the directed graph accessibility problem, since we have only to check that there are two paths from g to C's output-gate which join at some $\wedge$-gate in C, such that all gates on these paths evaluate to 1.

To prove completeness, we give a reduction from 3-SAT: Given a 3-CNF formula F, we construct an $\{\wedge, \vee\}$ circuit C_F with the same structure (that is, C_F actually is a formula). Let C' be some fixed circuit with differing numbers of accepting subtrees and subcircuits. Then the circuit $C =_{\text{def}} C_F \wedge C'$ will have $\#C(x) \neq \#_c C(x)$ for some input x if and only if F is satisfiable.

2. Observe that the algorithm given in the proof of case 1 is deterministic, once the input x is guessed. Hence the problem examined here is in P. Completeness is also shown along the lines above, this time using the P-complete circuit value problem. □

Remark 2.3. If we restrict our attention to circuits of depth d, then the problem in case 2 of the above lemma is in DSPACE(d); this is witnessed by the well-known depth-first search circuit evaluation algorithm which is used to show (uniform-)DEPTH(s) $\subseteq$ DSPACE(s).

2.3. Counting proof-circuits is #P-complete. For functions f, h, we say that $f \leq^{\log}_{1\text{-T}} h$ if there are functions g_1, g_2 computable in logarithmic space such that, for all x, $f(x) = g_1\big(x, h(g_2(x))\big)$.

THEOREM 2.4. PC *is complete for* #P *under* $\leq^{\log}_{1\text{-T}}$ *but not under* $\leq^{\text{p}}_m$ *unless* P = NP.

Proof. For a conjunctive normal form formula H with three literals per clause, define SA(H) to be the number of satisfying assignments to the variables of H. It is known (see [22]) that SA is $\leq^{\text{p}}_m$-complete for #P. We describe a $\leq^{\log}_{1\text{-T}}$-reduction from SA to PC. Let PC(C, x) denote the number of accepting subcircuits of circuit C with input x.

Let $H(x_1, \ldots, x_n) = \bigwedge_{i=1}^{m} C_i$ be a conjunctive normal form formula with three literals per clause. We define a $\{\vee, \wedge\}$-circuit C_H, all of whose inputs are set to 1, which has six levels and is stratified; i.e., the edges are only between gates of adjacent levels. The levels are as follows:

1. Level 1 consists of the input-gates v_{ikl} for $i = 1, \ldots, m$, $k = 0, \ldots, n$, and $l = 0, 1$. In the application below, all these will be set to 1.

2. Level 2 consists of the $\vee$-gates v_{ik} for $i = 1, \ldots, m$ and $k = 0, \ldots, n$. Gate v_{ik} has incoming edges from v_{ik0} and v_{ik1}.

3. Level 3 consists of the $\wedge$-gates v_i for $i = 1, \ldots, m$. Gate v_i has incoming edges from $v_{i0}, v_{i1}, \ldots, v_{in}$.

4. Level 4 consists of the $\wedge$-gates $w_1, \ldots, w_n, w'_1, \ldots, w'_n$. Gate w_j has incoming edges from all gates v_i such that x_j appears in the clause C_i. Gate w'_j has incoming edges from all gates v_i such that $\neg x_j$ appears in the clause C_i. Here, we stipulate that an $\wedge$-gate with no input wires computes the constant 1.

5. Level 5 consists of the $\vee$-gates $u_1, \ldots, u_n$. Gate u_j has incoming edges from the gates w_j and w'_j.

6. Level 6 consists of the output-gate which is an $\wedge$-gate. It has incoming edges from the gates $u_1, \ldots, u_n$.

Now let $a_1, \ldots, a_n \in \{0, 1\}$. We define the circuit $C_H^{[a_1 \cdots a_n]}$ to be that subcircuit of C_H which results from cutting the edge between w'_j and u_j if $a_j = 1$ and between w_j and u_j if $a_j = 0$ ($j = 1, \ldots, n$). The following facts are easy to see:

(i) $\mathrm{PC}(C_H, (1 \cdots 1)) = \sum_{a_1,\ldots,a_n \in \{0,1\}} \mathrm{PC}(C_H^{[a_1 \cdots a_n]}, (1 \cdots 1))$.

(ii) $(a_1, \ldots, a_n)$ satisfies exactly k clauses of H if and only if $\mathrm{PC}(C_H^{[a_1 \cdots a_n]}, (1 \cdots 1)) = 2^{k \cdot (n+1)}$.

(iii) If $(a_1, \ldots, a_n)$ satisfies H, then $\mathrm{PC}(C_H^{[a_1 \cdots a_n]}, (1 \cdots 1)) = 2^{m \cdot (n+1)}$, and if $(a_1, \ldots, a_n)$ does not satisfy H, then $\mathrm{PC}(C_H^{[a_1 \cdots a_n]}, (1 \cdots 1)) \leq 2^{(m-1) \cdot (n+1)}$.

(iv) $\mathrm{SA}(H) \cdot 2^{m \cdot (n+1)} \leq \mathrm{PC}(C_H, (1 \cdots 1)) < (\mathrm{SA}(H) + 1) \cdot 2^{m \cdot (n+1)}$.

Hence $\mathrm{SA}(H)$ can easily be computed from $\mathrm{PC}(C_H, (1 \cdots 1))$, i.e., $\mathrm{SA} \leq_{1\text{-T}}^{\log} \mathrm{PC}$.

Now, assume PC is complete for #P under $\leq_m^{\mathrm{p}}$ reductions. Let $A \in \mathrm{NP}$; then there is a function $f \in \#\mathrm{P}$ such that for all x, $x \in A \iff f(x) > 0$. Under our assumption, there is a many-one reduction g from f to PC; hence $x \in A \iff \mathrm{PC}\big(g(x)\big) > 0$. This latter condition, however, can be checked in polynomial time since the underlying Boolean circuit has a positive number of proof-circuits if and only if it evaluates to 1 (over the Boolean semiring). □

3. How to generate polynomials. A straight-line program P over variables $x_1, \ldots, x_m$ is a sequence of instructions of one of the following types: $x_i \leftarrow x_j + x_k$, $x_i \leftarrow x_j \cdot x_k$, $x_i \leftarrow 0$, $x_i \leftarrow 1$, where $j, k < i$. Every variable appears at most once on the left-hand side of the $\leftarrow$. Those variables that never appear on the left-hand side of the $\leftarrow$ are the *input variables*. The variable x_m is the *output variable*. Given values for the input variables, the values of all other variables are computed in the obvious way. The value computed by P is the value of the output variable. Let $p_P \colon \mathbb{N}^r \to \mathbb{N}$ denote the function computed in this way by P, where r is the number of input variables.

A straight-line program hence is just another way of looking at an arithmetic circuit. By the connection mentioned above between the problem of counting proof-trees and the evaluation of arithmetic circuits, we see that an obvious algorithm to determine the number of proof-trees of a circuit consists of producing the appropriate straight-line program and evaluating it in the order of its variables.

In section 1.1 we sketched an algorithm to count proof-circuits. We now give a more precise description of this algorithm: Given circuit C with input-gates $x_1, \ldots, x_n$ and inputs $a_1, \ldots, a_n \in \{0, 1\}$, let $g_1, \ldots, g_s$ be the non–input-gates of C in topological order (i.e., a total order on the gates of C which is consistent with the partial order given by the wires; hence $u < v$ whenever there is a wire from u to v).

1. Transform C into a straight-line program P as above.
2. In the following, we manipulate a polynomial p, which has variables $g_1, \ldots, g_s$, $x_1, \ldots, x_n$. Initially we set p to g_s.
3. **for** $i = s, s-1, \ldots, 1$, **do**
   1. Replace variable g_i in p by the right-hand side of that instruction in P, whose left-hand side is g_i.
   2. Transform p into the "sum-of-monomials" form.
   3. Reduce all powers of variables in p to 1; i.e., replace all z^2 by z, where z is a variable. At the end of this step, p will be a multilinear polynomial.
   4. Substitute the values of $a_1, \ldots, a_n$ for $x_1, \ldots, x_n$ and evaluate the resulting expression.

To see why this algorithm correctly counts proof-circuits, let $M(C)$ be the following nondeterministic Turing machine. On input x, $M(C)$ first evaluates all the gates of C on input x and prunes away any zero-gate. Unless the output-gate g_s of C was pruned away, $M(C)$ starts at g_s in the pruned circuit and implements a recursive procedure *evaluate*. At an $\wedge$-gate g, *evaluate* triggers consecutive recursive calls, one for each predecessor of g. At an $\vee$-gate, *evaluate* triggers a single recursive call for a

predecessor of g chosen nondeterministically. At a gate with no predecessor, *evaluate* simply accepts. It is well known that $M(C)$ has $\#C(x)$ accepting paths (see, e.g., [6]). But how can $M(C)$ be made to have $\#_c C(x)$ accepting paths instead? Simply by making sure that, whenever *evaluate* encounters a gate g a second time, g is treated as the constant gate 1: this will reflect the fact that the choice of an accepting subcircuit rooted at g was already made, on the first encounter! Let $M_c(C)$ be the machine $M(C)$ modified in this way. The algorithm of section 1.1, formalized above, precisely computes the number of accepting paths of $M_c(C)$. Encountering a gate g twice means that different paths out of g meet at an $\wedge$-gate (recall the proof of Lemma 2.2). This is equivalent, in the straight-line arithmetic program point of view, to obtaining a monomial in which g^2 appears. Replacing g^2 by g thus precisely models counting the number of accepting paths of $M_c(C)$, and thus the number of proof-circuits of C on input x.

To summarize, the algorithm to compute proof-circuits produces a polynomial (computing for argument $(x_1, \ldots, x_n)$ the number of proof-trees of circuit C on input $x_1 \cdots x_n$), which is obtained by evaluating the straight-line program given by C, but whenever possible during the computation we replace a power x^2 by x. This is only one type of a general class of *polynomial replacement systems* (PRSs), which we define below. PRSs will produce sets of polynomials from a given start polynomial, using rules replacing certain polynomials by other polynomials. This will be very similar to the way formal grammars produce sets of words from a start symbol, applying production rules.

In this paper we almost exclusively consider polynomials with nonnegative integer coefficients. This is motivated by the application to proof-trees and proof-circuits discussed above. We write $p(z_1, \ldots, z_s)$ to denote that p is such a polynomial in variables $z_1, \ldots, z_s$.

Below, the variable vector $\overline{x}$ will always be defined to consist of $\overline{x} = (x_1, \ldots, x_m)$. A variable x_i is *fictive* (or, *inessential*) in the polynomial $p(\overline{x})$ if for all $a_1, \ldots, a_m, a_i' \in \mathbb{N}$ we have $p(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_m) = p(a_1, \ldots, a_{i-1}, a_i', a_{i+1}, \ldots, a_m)$. This means that x_i is fictive in p if and only if p can be written as a term in which x_i does not appear.

DEFINITION 3.1. *A* polynomial replacement system (PRS) *is defined as a quadruple* $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$, *where*

(i) $\{x_1, \ldots, x_n\}$ *is the set of* terminal variables,

(ii) $\{x_{n+1}, \ldots, x_m\}$ *is the set of* nonterminal variables,

(iii) q *is a polynomial in the variables* $x_1, \ldots, x_m$, *the* start polynomial, *and*

(iv) R *is a finite set of* replacement rules, *i.e., a finite set of pairs of polynomials in the variables* $x_1, \ldots, x_m$.

How does such a system generate polynomials?

DEFINITION 3.2. *Let* $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ *be a PRS, and let* p_1, p_2 *be polynomials in the variables* $\overline{x}$.

$$p_1 \underset{S}{\Longrightarrow} p_2 \iff_{\text{def}} \text{there exist } (p_3, p_4) \in R \text{ and a polynomial } p_5(\overline{x}, y) \text{ such that } p_1(\overline{x}) = p_5(\overline{x}, p_3(\overline{x})) \text{ and } p_2(\overline{x}) = p_5(\overline{x}, p_4(\overline{x})).$$

Let $\overset{*}{\underset{S}{\Longrightarrow}}$ *be the reflexive and transitive closure of* $\underset{S}{\Longrightarrow}$, *i.e.,* $p_1 \overset{*}{\underset{S}{\Longrightarrow}} p_2$, *if and only if there exist* $t \geq 0$ *and polynomials* $q_0(\overline{x}), q_1, (\overline{x}) \ldots, q_t(\overline{x})$ *such that* $p_1 = q_0 \underset{S}{\Longrightarrow} q_1 \underset{S}{\Longrightarrow} \cdots \underset{S}{\Longrightarrow} q_t = p_2$.

It turns out that the above form for derivations can be simplified as follows.

DEFINITION 3.3. *Let S, p_1, p_2 be as above.*

$$p_1 \underset{S}{\rightarrow} p_2 \iff_{\text{def}} \text{there exist } (p_3, p_4) \in R \text{ and polynomials } p_5(\overline{x}), p_6(\overline{x}) \text{ such that } p_1(\overline{x}) = p_5(\overline{x}) \cdot p_3(\overline{x}) + p_6(\overline{x}) \text{ and } p_2(\overline{x}) = p_5(\overline{x}) \cdot p_4(\overline{x}) + p_6(\overline{x}).$$

Let $\overset{*}{\underset{S}{\rightarrow}}$ be the reflexive and transitive closure of $\underset{S}{\rightarrow}$.

LEMMA 3.4 (normal form of replacement). *For any PRS $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ and any polynomials $p_1(\overline{x}), p_2(\overline{x})$, we have*

$$p_1 \overset{*}{\underset{S}{\Longrightarrow}} p_2 \quad \text{if and only if} \quad p_1 \overset{*}{\underset{S}{\rightarrow}} p_2.$$

Proof. Clearly, $p_1 \underset{S}{\rightarrow} p_2$ implies $p_1 \underset{S}{\Longrightarrow} p_2$. We prove that $p_1 \underset{S}{\Longrightarrow} p_2$ implies $p_1 \overset{*}{\underset{S}{\rightarrow}} p_2$. Let $(p_3, p_4) \in R$, and let p_5 be a polynomial such that $p_1(\overline{x}) = p_5(\overline{x}, p_3(\overline{x}))$ and $p_2(\overline{x}) = p_5(\overline{x}, p_4(\overline{x}))$. We can represent p_5 as $p_5(\overline{x}, y) = \sum_{i=0}^{k} q_i(\overline{x}) \cdot y^i$ with suitable $k \geq 0$ and polynomials $q_0, q_1, \ldots, q_k$. Hence $p_1(\overline{x}) = \sum_{i=0}^{k} q_i(\overline{x}) \cdot p_3(\overline{x})^i$ and $p_2(\overline{x}) = \sum_{i=0}^{k} q_i(\overline{x}) \cdot p_4(\overline{x})^i$. Defining the polynomials

$$r_m(\overline{x}, y) =_{\text{def}} \sum_{i=0}^{m-1} q_i(\overline{x}) \cdot p_4(\overline{x})^i + \sum_{i=m}^{k} q_i(\overline{x}) \cdot p_4(\overline{x})^{m-1} \cdot p_3(\overline{x})^{i-m} \cdot y$$

for $m = 1, \ldots, k$, and

$$t_m(\overline{x}) =_{\text{def}} \sum_{i=0}^{m-1} q_i(\overline{x}) \cdot p_4(\overline{x})^i + \sum_{i=m}^{k} q_i(\overline{x}) \cdot p_4(\overline{x})^{m-1} \cdot p_3(\overline{x})^{i-m+1}$$

for $m = 1, \ldots, k+1$, we get $r_m(\overline{x}, p_3(\overline{x})) = t_m(\overline{x})$ and $r_m(\overline{x}, p_4(\overline{x})) = t_{m+1}(\overline{x})$ for $m = 1, \ldots, k$, and moreover, $t_1(\overline{x}) = p_1(\overline{x})$ and $t_{k+1}(\overline{x}) = p_2(\overline{x})$.

Consequently, $p_1 = t_1 \underset{S}{\rightarrow} t_2 \underset{S}{\rightarrow} \cdots \underset{S}{\rightarrow} t_k \underset{S}{\rightarrow} t_{k+1} = p_2$ and thus $p_1 \overset{*}{\underset{S}{\rightarrow}} p_2$. ∎

A PRS thus generates a set of polynomials; hence we have the following definition.

DEFINITION 3.5. *For a PRS $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$, let*

$$\text{POLY}(S) = \big\{ p(x_1, \ldots, x_n) \;\big|\; \text{there exists } p'(\overline{x}) \text{ such that } q \overset{*}{\underset{S}{\Longrightarrow}} p' \text{ and } p(x_1, \ldots, x_n) = p'(x_1, \ldots, x_n, a_{n+1}, \ldots, a_m) \text{ for all } a_{n+1}, \ldots, a_m \in \mathbb{N} \big\}.$$

From the set $\text{POLY}(S)$ of polynomials we derive several sets of natural numbers, whose complexities we will determine in the upcoming sections.

DEFINITION 3.6. *Let $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ be a PRS. Define*

(i) $\text{RANGE}(S) =_{\text{def}} \big\{ p(a) \;\big|\; p \in \text{POLY}(S) \wedge a \in \mathbb{N}^n \big\}$;

(ii) $\text{EVAL}(S) =_{\text{def}} \big\{ (a, p(a)) \;\big|\; p \in \text{POLY}(S) \text{ and } a \in \mathbb{N}^n \big\}$.

Observe that if we also allow negative numbers as coefficients for our polynomials, then there are PRSs S such that $\text{RANGE}(S)$ is not decidable. This is seen as follows. By the Robinson–Matiyasevich result (see [15]), every recursively enumerable set can be represented in the form $\big\{ p(a) \;\big|\; a \in \mathbb{N}^n \big\}$, where p is a suitable n-ary polynomial with integer coefficients. Now let p be such an n-ary polynomial such that $\big\{ p(a) \;\big|\; a \in \mathbb{N}^n \big\}$ is not decidable. Defining the PRS $S_p =_{\text{def}} (\{x_1, \ldots, x_n\}, \emptyset, p, \emptyset)$, we obtain $\text{POLY}(S_p) = \{p\}$ and $\text{RANGE}(S_p) = \big\{ p(a) \;\big|\; a \in \mathbb{N}^n \big\}$.

Besides the membership problems POLY(S), RANGE(S), and EVAL(S), we also consider the corresponding *variable membership problems*.

DEFINITION 3.7.

(i) POLY$(\cdot) =_{\text{def}} \{(S,p)|\ S$ *PRS and* $p \in$ POLY$(S)\}$;
(ii) RANGE$(\cdot) =_{\text{def}} \{(S,a)|\ S$ *PRS and* $a \in$ RANGE$(S)\}$;
(iii) EVAL$(\cdot) =_{\text{def}} \{(S,a,p(a))|\ S$ *PRS*, $p \in$ POLY(S), *and* $a \in \mathbb{N}^*\}$.

4. Representations of polynomials and equivalence test. To discuss the complexity of the sets defined above we have to talk about representations of polynomials. We distinguish different kinds of representations.

DEFINITION 4.1. *Let* $p(\overline{x})$ *be a polynomial.*

1. *The* full representation *of* p *is its sum-of-monomials form. That is, we describe* p *as a sequence of vectors* $(c, e_1, \ldots, e_m)$*, each consisting of* $m+1$ *nonnegative integers, where* $c, e_1, \ldots, e_m$ *are given in unary. Such a vector stands for the monomial* $c \cdot \prod_{i=1}^{m} x_i^{e_i}$.

2. *In the* extended full representation (ef representation)*, we describe* p *as a sequence of vectors as above, but this time* c *is given in binary while all numbers* $e_1, \ldots, e_m$ *are given in unary.*

3. *In the* formula representation*,* p *is described by a formula involving the variables* $\overline{x}$*. More precisely, the set of formulas in variables* $\overline{x}$ *is defined inductively by the following rules:*

1. $x_1, \ldots, x_m, 0, 1$ *are formulas.*
2. *If* F, G *are formulas, then so are* $(F+G)$ *and* $(F \times G)$.

3. *In the* straight-line program representation (slp representation)*, we describe* p *by a straight-line program with input variables* $\overline{x}$ *that computes* p.

If Φ is a representation of a polynomial of one of the above types, then we denote the polynomial represented by Φ by p_Φ.

It is easy to see that the above definition introduces a chain of representations with increasing succinctness. For every representation of type (i) of a polynomial, equivalent representations of types (j) for $j > i$ can be computed in logarithmic space. Moreover, if p has a full or ef or formula representation of size n, then the degree of p is bounded by n, and if p has an slp representation of size n, then the degree of p is at most exponential in n.

The idea to use slp representations as a data structure for polynomials was introduced and promoted by Erich Kaltofen (see, e.g., [12]).

In the upcoming sections we will have to determine if two representations stand for the same polynomial. This gives rise to the *equivalence problem*

$$\left\{ \langle \Phi_1, \Phi_2 \rangle \;\middle|\; \Phi_1 \text{ and } \Phi_2 \text{ are representations of polynomials such that } p_{\Phi_1} = p_{\Phi_2} \right\}$$

for the various types of representations. A result similar to the next theorem was obtained in [10].

THEOREM 4.2. *The equivalence problem is*

1. *in* P *for full and ef representation;*
2. *in* coRP *for formula and slp representation.*

The proof will make instrumental use of the following well-known result [18, 31] (see also [16, p. 165f]).

PROPOSITION 4.3 (Schwartz–Zippel). *Let* $p(x_1, \ldots, x_n)$ *be a polynomial of degree* d *over some field* $\mathbb{F}$ *that is not the zero polynomial. Let* $\mathbb{S} \subseteq \mathbb{F}$ *be a finite set, and let*

$a_1, \ldots, a_n$ be chosen independently and uniformly at random from $\mathbb{S}$. Then

$$\operatorname{prob}[p(a_1, \ldots, a_n) = 0] \leq \frac{d}{|\mathbb{S}|}.$$

Proof of Theorem 4.2. Two polynomials in (extended) full representation are equivalent if and only if they have the same monomials. This can be checked in polynomial time.

Given now two formulas F_1 and F_2, we know that the degree d of the two represented polynomials is at most linear in the length of F_1 and F_2. Hence by Proposition 4.3 it suffices to evaluate F_1 and F_2 for a random input drawn from an exponential number of possible input vectors. The actual evaluation of the formulas is in P.

Finally, we are given two straight-line programs P_1 and P_2. Compared with the above, we now have to deal with the fact the degree of the represented polynomial can be exponential; hence the numbers we operate with can be double exponential in $n =_{\text{def}} |P_1| + |P_2|$. Therefore, we cannot carry out the necessary arithmetic operations in polynomial time. Instead we proceed as follows (see also [16, p. 169]).

Suppose the two numbers $a, b < 2^{2^n}$ are different; then $|a - b|$ can have at most 2^n different prime divisors. Let $\pi(m)$ denote the number of primes smaller than or equal to m. Hence for a random prime smaller than $k2^n \log(k2^n)$, the probability that $a \equiv b \pmod{p}$ is at most $\frac{2^n}{\pi(k2^n \log(k2^n))} \leq O(\frac{1}{k})$. Thus if two numbers are different, then they are different modulo most primes having a polynomial number of bits.

Hence P_1 and P_2 are not equivalent if and only if for most inputs within an exponential range we get inequality if and only if for most inputs within an exponential range we get inequivalence modulo most primes having a polynomial number of bits. Since the set of prime numbers is in ZPP, the resulting algorithm is an RP algorithm. □

5. Different types of replacement systems. The definition of PRSs we presented above is very general. Here, we introduce a number of natural restrictions. Our approach is similar to the way different restrictions of grammar types were introduced, e.g., in the definition of the classes of the Chomsky hierarchy. We will later view the problems of counting proof-trees and proof-circuits as two instances of a problem about these restricted PRS types.

5.1. Simple polynomial replacement systems.

DEFINITION 5.1. *A PRS $S = (\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R)$ is* simple *(or* context-free*), if the polynomials in the left-hand sides of the rules of R are variables from $\{x_{n+1}, \ldots, x_m\}$.*

All definitions made in the preceding section for general PRSs carry over to the special cases of simple systems. However, for a simple PRS we additionally define a particular type of replacement, where the application of a rule (z, q) results in the replacement of *all occurrences* of z with q. This latter form is denoted by $\underset{S}{\models\!\!\Rightarrow}$, in contrast to the notation $\underset{S}{\Longrightarrow}$ for the derivations defined so far. Formally, we have the following.

DEFINITION 5.2. *Let $S = (\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R)$ be a simple PRS.*

$$p_1 \underset{S}{\models\!\!\Rightarrow} p_2 \iff_{\text{def}} \text{there exist } (x_i, p_3) \in R \text{ such that } p_2(\overline{x}) = p_1(x_1, \ldots, x_{i-1}, p_3(\overline{x}), x_{i+1}, \ldots, x_m).$$

Let $\underset{S}{\overset{*}{\models\!\!\Rightarrow}}$ be the reflexive and transitive closure of $\underset{S}{\models\!\!\Rightarrow}$.

For the sets of polynomials and numbers derived from simple systems using our new derivation type, we use the same names as before but now use square brackets $[\cdots]$ instead of parentheses $(\cdots)$; formally we have the following.

DEFINITION 5.3. *For a simple PRS* $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$, *let*

$$\text{POLY}[S] = \Big\{ p(x_1, \ldots, x_n) \;\Big|\; \text{there exists } p'(\overline{x}) \text{ such that } q \overset{*}{\underset{S}{\Longrightarrow}} p' \text{ and } p(x_1, \ldots, x_n) = p'(x_1, \ldots, x_n, a_{n+1}, \ldots, a_m) \text{ for all } a_{n+1}, \ldots, a_m \in \mathbb{N} \Big\}.$$

As in the case of $\text{POLY}(S)$, we have the following definition.

DEFINITION 5.4. *Let* $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ *be a simple PRS. Define*

(i) $\text{RANGE}[S] =_{\text{def}} \big\{\, p(a) \;\big|\; p \in \text{POLY}[S] \wedge a \in \mathbb{N}^n \,\big\}$;

(ii) $\text{EVAL}[S] =_{\text{def}} \big\{\, (a, p(a)) \;\big|\; p \in \text{POLY}[S] \text{ and } a \in \mathbb{N}^n \,\big\}$.

We also define the following variable membership problems:

(i) $\text{POLY}[\cdot] =_{\text{def}} \big\{\, (S, p) \;\big|\; S \text{ simple PRS and } p \in \text{POLY}[S] \,\big\}$;

(ii) $\text{RANGE}[\cdot] =_{\text{def}} \big\{\, (S, a) \;\big|\; S \text{ simple PRS and } a \in \text{RANGE}[S] \,\big\}$;

(iii) $\text{EVAL}[\cdot] =_{\text{def}} \big\{\, (S, a, p(a)) \;\big|\; S \text{ simple PRS}, p \in \text{POLY}[S] \text{ and } a \in \mathbb{N}^* \,\big\}$.

It is clear that for any simple PRS S, we have $\text{POLY}[S] \subseteq \text{POLY}(S)$; hence we also obtain $\text{RANGE}[S] \subseteq \text{RANGE}(S)$, $\text{EVAL}[S] \subseteq \text{EVAL}(S)$, $\text{POLY}[\cdot] \subseteq \text{POLY}(\cdot)$, $\text{RANGE}[\cdot] \subseteq \text{RANGE}(\cdot)$, and $\text{EVAL}[\cdot] \subseteq \text{EVAL}(\cdot)$.

5.2. Simple deterministic or acyclic PRSs.

DEFINITION 5.5. *A PRS* $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ *is said to be* deterministic *if no two different rules in* R *have the same left-hand side.*

DEFINITION 5.6. *Let* $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ *be a PRS. The* dependency graph G_S *of* S *is the directed graph* $G_S = \big(\{1, \ldots, m\}, E_S\big)$, *where* E_S *consists of all edges* (j, i) *for which there exists a rule* $(p_1, p_2) \in R$ *such that* x_i *is essential in* p_1 *and* x_j *is essential in* p_2. *The PRS* S *is said to be* acyclic *if its dependency graph* G_S *is acyclic.*

LEMMA 5.7. *For every simple and deterministic PRS* S, *there exists a simple, deterministic, and acyclic PRS* S' *such that* $\text{POLY}(S) = \text{POLY}(S')$ *and* $\text{POLY}[S] = \text{POLY}[S']$. *The system* S' *can be obtained from* S *in polynomial time.*

Proof. Let $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ be a simple and deterministic PRS. If S is not acyclic, then there exist $k \geq 1$, $i_1, \ldots, i_k \in \{n+1, \ldots, m\}$, and polynomials $p_1(\overline{x}), \ldots, p_k(\overline{x})$ such that $(x_{i_1}, p_1), (x_{i_2}, p_2), \ldots, (x_{i_k}, p_k) \in R$ and $x_{i_{r+1}}$ is essential in p_r for $r = 1, \ldots, k$ (where we set $i_{k+1} = i_1$). Since $x_{i_1}, \ldots, x_{i_k}$ are nonterminal variables, we claim that they have to be fictive in every polynomial p such that $q \overset{*}{\underset{S}{\Longrightarrow}} p \overset{*}{\underset{S}{\Longrightarrow}} p'$ and $p' \in \text{POLY}(S)$. Indeed, S is deterministic, and hence such a replacement would necessarily run along the cycle described by the rules $(x_{i_1}, p_1), (x_{i_2}, p_2), \ldots, (x_{i_k}, p_k)$. Hence, if p has a nonfictive variable from $\{x_{i_1}, \ldots, x_{i_k}\}$, then every p' such that $p \overset{*}{\underset{S}{\Longrightarrow}} p'$ will have a nonfictive variable from $\{x_{i_1}, \ldots, x_{i_k}\}$. Thus we have $\text{POLY}(S) = \text{POLY}(S')$ and $\text{POLY}[S] = \text{POLY}[S']$, where

$$S' =_{\text{def}} \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\} \setminus \{x_{i_1}, \ldots, x_{i_k}\}, q, R'\big) \quad \text{and}$$

$$R' =_{\text{def}} R \setminus \big\{\, (x_i, p) \;\big|\; x_i \text{ or an essential variable in } p \text{ is from } \{x_{i_1}, \ldots, x_{i_k}\} \,\big\}.$$

Now we repeat this cycle removement step as long as the PRS still has cycles. Since in every step at least one nonterminal variable is removed, we get after a polynomial number of steps a PRS which is simple, deterministic, and acyclic. ∎

We also obtain the following easy properties.

LEMMA 5.8.

1. *If S is a simple and deterministic PRS, then* $\text{POLY}(S) = \text{POLY}[S]$*, and this set consists of at most one polynomial.*

2. *If S is a simple and acyclic PRS, then* $\text{POLY}(S)$ *and* $\text{POLY}[S]$ *are finite.*

Proof. If S is simple, deterministic, and without loss of generality, moreover, acyclic, then the rules replace every variable by a unique polynomial. Since S is simple, we always have to replace all occurrences of each nonterminal variable sooner or later. Hence the result of the whole derivation process is unique.

If S is acyclic, then every nonterminal variable can only be replaced in a bounded number of ways, and hence we can only obtain finitely many polynomials. □

Note that there are simple and acyclic PRSs S such that $\text{POLY}[S] \subsetneq \text{POLY}(S)$. For example take $S = (\{x\}, \{z\}, 2z, \{(z, x), (z, 2x)\})$, where $\text{POLY}[S] = \{2x, 4x\}$ and $\text{POLY}(S) = \{2x, 3x, 4x\}$. Thus, the requirement that S is deterministic is necessary in Lemma 5.8.1.

In the remainder of this subsection, we relate simple deterministic and simple acyclic PRSs to different forms of circuits operating over the natural numbers.

First, it is intuitively clear that there is some connection between simple, deterministic, and acyclic systems and straight-line programs. This is made precise in the following lemma.

LEMMA 5.9.

1. *If S is a simple, deterministic, and acyclic PRS such that* $\text{POLY}(S) \neq \emptyset$*, then there exists a straight-line program P such that* $\text{POLY}(S) = \{p_P\}$.

2. *If P is a straight-line program, then there exists a simple, deterministic, and acyclic PRS S such that* $\{p_P\} = \text{POLY}(S)$.

3. *The transformations from a simple, deterministic, and acyclic PRS to the corresponding straight-line program and vice-versa can be computed in logarithmic space.*

Proof. The program P is obtained from S by transforming every single replacement rule into a sequence of straight-line program instructions and then ordering these according to a topological order of G_S. Statement 2 is proved similarly. Statement 3 is obvious. □

Next we show that acyclic systems are strongly related to a new type of arithmetic circuit we now define. These circuits are immediate generalizations of *integer expressions*, introduced by Stockmeyer and Meyer [19]. Therefore, we call our circuits *integer circuits* (not to be confused with ordinary arithmetic circuits), or, referring to the operations allowed, $(\cup, +, \times)$-circuits.

An integer circuit with n inputs is a circuit C where the inner nodes compute one of the operations $\cup, +, \times$. Such a circuit C has a specified output-gate g_s. It computes a function $f_C \colon \mathbb{N}^n \to 2^{\mathbb{N}}$ as follows: We first define for every gate $g \in C$ the function f_g computed by g.

1. If g is an input-gate x_i, then $f_g(a_1, \ldots, a_n) = \{a_i\}$ for all $a_1, \ldots, a_n \in \mathbb{N}$.
2. If g is a $+$-gate with predecessors g_l, g_r, then $f_g(a_1, \ldots, a_n) = \{\, k + m \mid k \in f_{g_l}(a_1, \ldots, a_n), m \in f_{g_r}(a_1, \ldots, a_n) \,\}$. The function computed by a $\times$-gate is defined analogously.
3. If g is a $\cup$-gate with predecessors g_l, g_r, then $f_g(a_1, \ldots, a_n) = f_{g_l}(a_1, \ldots, a_n) \cup f_{g_r}(a_1, \ldots, a_n)$.

Finally, the function computed by C is $f_C =_{\text{def}} f_{g_s}$.

The following relation between simple, acyclic replacement systems and integer

circuits is obtained by an easy induction.

LEMMA 5.10.

1. *For every simple, acyclic PRS* $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$, *there is an integer circuit* C *with* n *inputs such that* $f_C(a) = \big\{\, b \;\big|\; (a,b) \in \text{EVAL}(S) \,\big\}$ *for all* $a \in \mathbb{N}^n$.

2. *For every integer circuit* C *with* n *inputs, there is a simple, acyclic PRS* S *such that* $\big\{\, b \;\big|\; (a,b) \in \text{EVAL}(S) \,\big\} = f_C(a)$ *for all* $a \in \mathbb{N}^n$.

3. *The transformations from a simple, acyclic PRS to the corresponding integer circuit and vice-versa can be computed in logarithmic space.*

We consider the following problems:

$$\mathbb{N}\text{-MEMBER}(\cup,+,\times) =_{\text{def}} \big\{\, (C, a_1, \ldots, a_n, b) \;\big|\; C \text{ is an integer circuit with } n \text{ inputs}, \; a_1, \ldots, a_n, b \in \mathbb{N} \text{ and } b \in f_C(a_1, \ldots, a_n) \,\big\},$$

$$\mathbb{N}\text{-RANGE}(\cup,+,\times) =_{\text{def}} \big\{\, (C, b) \;\big|\; C \text{ is an integer circuit with } n \text{ inputs}, \; b \in \mathbb{N} \text{ and } (\exists a_1, \ldots, a_n) b \in f_C(a_1, \ldots, a_n) \,\big\}.$$

Analogous notation will be used when we restrict the gate types allowed.

The following lemma is immediate from Lemma 5.10.

LEMMA 5.11. *For all representations, the following hold:*

1. $\mathbb{N}\text{-MEMBER}(\cup,+,\times) \equiv_m^{\log} \text{EVAL}(\cdot)$.
2. $\mathbb{N}\text{-RANGE}(\cup,+,\times) \equiv_m^{\log} \text{RANGE}(\cdot)$.

5.3. Idempotent polynomial replacement systems.

DEFINITION 5.12. *For a PRS* $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$, *let* $S_{\text{idem}} =_{\text{def}} \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R \cup \big\{\, (x_i^2, x_i) \;\big|\; 1 \le i \le m \,\big\}\big)$ *be the* idempotent PRS *derived from* S.

In the case that S is simple (deterministic, acyclic, respectively), we will say that S_{idem} is an idempotent simple (idempotent deterministic, idempotent acyclic, respectively) PRS. Note that if S_{idem} is idempotent simple, we mean that it is an idempotent PRS that originates from a simple PRS, not that S_{idem} itself is simple (similarly for deterministic and acyclic systems).

For a PRS $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ and $a \in \mathbb{N}^n$, we use the notation $\min \text{EVAL}(S, a)$ as shorthand for $\min\big\{\, p(a) \;\big|\; p \in \text{POLY}(S) \,\big\}$ (analogously, we use $\max \text{EVAL}(S, a)$).

LEMMA 5.13.

1. *For every Boolean circuit* C, *input* x, *and* $k \in \mathbb{N}$, *there exists a simple, deterministic, and acyclic PRS* S *such that*

$$\min \text{EVAL}\big(S_{\text{idem}}, (1, \ldots, 1)\big) = \#_c C(x), \text{ and } \max \text{EVAL}\big(S_{\text{idem}}, (1, \ldots, 1)\big) = \#C(x).$$

2. *For every simple, deterministic, and acyclic PRS* S_{idem}, *there exists a Boolean circuit* C *such that*

$$\min \text{EVAL}\big(S_{\text{idem}}, (1, \ldots, 1)\big) = \#_c C(x), \text{ and } \max \text{EVAL}\big(S_{\text{idem}}, (1, \ldots, 1)\big) = \#C(x).$$

3. *The transformations from an idempotent simple, deterministic, and acyclic PRS to the corresponding circuit and vice-versa can be computed in logarithmic space.*

Proof. Arithmetic circuits and straight-line programs are only two different views of the same concept. Hence we go from circuit C to system S and vice versa exactly as in Lemma 5.9. Clearly the simple, deterministic, and acyclic system S computes $\#C(x)$ as explained in the discussion in the beginning of section 3.

Now convert S into the idempotent system S_{idem}. That the number of proof-circuits coincides with the minimal element in $\text{EVAL}\big(S_{\text{idem}}, (1, \ldots, 1)\big)$ is a consequence of the algorithm given at the beginning of section 3. The minimal element is obtained if and only if during the derivation of a polynomial all rules (x_i^2, x_i) are applied whenever possible. The maximal element is obtained if and only if we never pick such a rule, i.e., if we use only rules from S. □

6. Complexity results for simple replacement systems.

6.1. Deterministic systems. In this section, we consider the complexity of the above defined sets for simple replacement systems. Let us start with the complexity of fixed membership problems.

THEOREM 6.1. *Let S be simple and deterministic.*

1. $\text{POLY}(S), \text{POLY}[S] \in \text{P}$ *for all representations, and* $\text{POLY}(S), \text{POLY}[S]$ *are* P*-complete for the slp representation.*

2. $\text{RANGE}(S), \text{RANGE}[S] \in \text{NP}$ *for all representations. In fact, for all representations there are systems S such that the problems* $\text{RANGE}(S)$ *and* $\text{RANGE}[S]$ *are* NP*-complete.*

3. $\text{EVAL}(S), \text{EVAL}[S] \in \text{TC}^0$ *for all representations.*

Proof. First we recall that $\text{POLY}(S)$ and $\text{POLY}[S]$ consist of at most one polynomial p. In full and ef representation, p has only finitely many representations; hence the complexity of $\text{POLY}(S), \text{POLY}[S]$ is trivial. However, in formula and slp representation, p may have infinitely many representations. That is, we face the following problem: Given a straight-line program P (the case of formulas is even easier), does $p_P = p$ (where p is fixed)?

Say that P is reduced if the following holds: P has no instructions of the form $x_j \leftarrow \cdots$, where the x_j never appears on the right-hand side of any other instructions (unless x_j is the output variable). Additionally, if $x_j \leftarrow x_i + x_k$ is an instruction in P, then the polynomials computed by x_i and x_k are both not the constant zero polynomial (otherwise, the instruction is useless, for, e.g., if $x_i \equiv 0$, then $x_j \equiv x_k$, and hence we do not compute anything new; we might delete the above instruction and in further instructions we use x_k instead of x_j). Similarly, for $x_j \leftarrow x_i \cdot x_k$ we require that x_i and x_k are not constant one.

Now the following holds: Every polynomial p has only finitely many representations by reduced straight-line programs (except for an isomorphic renumbering of the variables). Moreover, a program P can be transformed into reduced form as follows: Inductively, determine the sets of variables that compute the constant zero or constant one polynomial. Then remove those instructions that do not compute a new polynomial, and change the variables in the other instructions as described above. This shows that $\text{POLY}(S), \text{POLY}[S] \in \text{P}$.

Hardness follows from a reduction from the circuit value problem as follows: Let C be a Boolean circuit. Let p be the polynomial that gives the number of proof-trees of C. Let $\text{POLY}(S) = \text{POLY}[S] = \{p'\}$; then $p + p' \in \text{POLY}(S) = \text{POLY}[S]$ iff C does not evaluate to 1 and if and only if C is not in CVP.

For statement 2 we have to decide the range of a multivariate polynomial with nonnegative coefficients. This is clearly in NP by the obvious guessing algorithm. For the hardness proof, we give a reduction from the quadratic Diophantine equations problem (problem AN8 in [7, p. 250]) as follows: This problem consists of all triples (a, b, c) such that the quadratic equation $ax^2 + by = c$ has a solution in positive integers. Define $\langle x, y, z\rangle =_{\text{def}} z + (y+z)^2 + (x+y+z)^3$. Observe that $\langle \cdots \rangle$ is one-one. Now, define the four-variable polynomial $p(u, v, x, y) =_{\text{def}} ux^2 + vy$. Then for all

$a, b, c \in \mathbb{N}$, the equation $ax^2 + by = c$ has a solution if and only if there are x, y such that $p(a, b, x, y) = c$, if and only if there are x, y such that $\langle a, b, p(a, b, x, y)\rangle = \langle a, b, c\rangle$, and if and only if there are u, v, x, y such that $\langle u, v, p(u, v, x, y)\rangle = \langle a, b, c\rangle$. Hence we see that the quadratic Diophantine equations problem reduces to the range of the polynomial $q(u, v, x, z) =_{\text{def}} \langle u, v, p(u, v, x, y)\rangle$.

For statement 3 observe that we have to evaluate a fixed polynomial; hence the result follows since addition and multiplication are in TC^0. □

Concerning variable membership problems of simple, deterministic systems, we obtain the following theorem.

THEOREM 6.2. *For simple and deterministic PRSs and for all representations,*

1. POLY$(\cdot)$ *and* POLY$[\cdot]$ *are in* coRP *and* P*-hard under* $\leq_m^{\log}$ *(in fact* P*-complete for full and ef representations)*;
2. RANGE$(\cdot)$ *and* RANGE$[\cdot]$ *are* NP*-complete under* $\leq_m^{\log}$;
3. EVAL$(\cdot)$ *and* EVAL$[\cdot]$ *are* P*-complete under* $\leq_m^{\log}$.

Proof. Containment. In all cases, given the simple and deterministic PRS S, first construct in polynomial time the slp representation of the unique polynomial p (if any) in POLY(S) = POLY$[S]$ by Lemmas 5.7 and 5.8. Statement 1 then follows by equivalence test (Theorem 4.2). For statement 2, guess suitable input values and evaluate p; for statement 3, just evaluate p (until finished or until the bounds are exceeded, as above).

Hardness: Statement 3. We give a reduction from the circuit value problem CVP as follows: Given a circuit C and an input a to C, produce the replacement system S which computes the number of proof-trees of C on input a. S is simple and deterministic (see section3), and if we "hardwire" the input a of C directly into S, the resulting polynomial has only fictive variables, i.e., is essentially a natural number. Now C accepts a if and only if this number is greater than 0; hence $(C, a) \in$ CVP if and only if $(S, a, 0) \notin$ EVAL$[\cdot]$ = EVAL$(\cdot)$. Statement 1: Proceeding as above, we obtain $(C, a) \in$ CVP if and only if the constant 0 polynomial is not in POLY(S) = POLY$[S]$. Statement 2: This follows from Theorem 6.1.2. □

6.2. Acyclic systems. Let us next deal with simple, acyclic, but not necessarily deterministic systems. For such systems S the sets POLY(S) and POLY$[S]$ are finite (by Lemma 5.8); hence we obtain the following analogous to Theorem 6.1.

THEOREM 6.3. *Let S be simple and acyclic. For all representations the following hold:*

1. POLY(S), POLY$[S] \in$ P.
2. RANGE(S), RANGE$[S] \in$ NP*, and there are systems S such that the problems* RANGE(S) *and* RANGE$[S]$ *are* NP*-complete.*
3. EVAL(S), EVAL$[S] \in \text{TC}^0$.

Again, interesting questions arise when we examine variable membership problems.

THEOREM 6.4. *For simple and acyclic PRSs and for all representations,*

1. POLY$[\cdot]$ *is contained in* MA *and is* NP*-hard under* $\leq_m^{\log}$,
2. RANGE$[\cdot]$ *and* EVAL$[\cdot]$ *are* NP*-complete under* $\leq_m^{\log}$.

Proof. The containment proofs are similar to that of Theorem 6.2, but before applying Lemma 5.7 we select nondeterministically a deterministic PRS by deleting some of the rules of the originally given PRS. The claim follows since MA $= \exists \cdot$ coRP [8].

Hardness for POLY$[\cdot]$ is proven by giving a logspace many-one reduction from the sum-of-subset problem SOS (problem SP13 in [7, p. 223]) to POLY$[\cdot]$ as follows: Let

$m, b_1, \ldots, b_m, c \in \mathbb{N} \setminus \{0\}$. Define the PRS S by

$$S =_{\text{def}} \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_{n+m}\}, x_{n+m}, R\big), \text{ where}$$
$$R =_{\text{def}} \big\{(x_{n+1}, 0), (x_{n+1}, b_1)\big\} \cup \bigcup_{i=2}^{m} \big\{(x_{n+i}, x_{n+i-1}), (x_{n+i}, x_{n+i-1} + b_i)\big\}.$$

Note that S is acyclic and simple, and $\text{POLY}[S]$ consists only of polynomials without essential variables. For $c \in \mathbb{N}$, define $p_c(x_1, \ldots, x_n) =_{\text{def}} c$. Now we obtain

$$(b_1, \ldots, b_m, c) \in \text{SOS} \iff p_c \in \text{POLY}[S] \iff (S, p_c) \in \text{POLY}[\cdot].$$

Since a deterministic PRS can (by Lemma 5.7) be assumed to be acyclic; hardness for $\text{RANGE}[\cdot]$ follows as in Theorem 6.2.

We next prove hardness for $\text{EVAL}[\cdot]$ by giving a logspace many-one reduction from the SOS problem: Given $m, b_1, \ldots, b_m, c > 0$, define S as in the proof of $\text{POLY}[\cdot]$ above. Then we obtain $(b_1, \ldots, b_m, c) \in \text{SOS} \iff (S, a, c) \in \text{EVAL}[\cdot]$ for any $a \in \mathbb{N}^n$. □

Next, we turn to different variable membership problems for simple, acyclic systems under "$\Longrightarrow$" derivations.

Stockmeyer and Meyer considered integer expressions (in our terminology, these are integer circuits with fan-out of non–input-gates at most 1) where the only operations allowed are $\cup$ and $+$. They proved that the membership problem in that case is NP-complete. It is easy to see that their result carries over to the case in which we also allow multiplication; i.e., the problems $\mathbb{N}$-MEMBER$(\cup, +)$ and $\mathbb{N}$-MEMBER$(\cup, +, \times)$ for expressions are NP-complete. The corresponding problems for circuits were not considered in their paper but in later papers by Wagner [28, 29] (under the name *hierarchical descriptions*). Only PSPACE as an upper bound is known from there, but recently it was shown by Ke Yang that $\mathbb{N}$-MEMBER$(\cup, +, \times)$ is PSPACE-hard [30]. (We remark that the problem $\mathbb{N}$-MEMBER$(\cup, +)$, which is not so interesting from a PRS point of view, can be shown to be NP-complete also in the circuit case.)

By Lemma 5.11 the member and range problems for these circuits are equivalent to the $\text{EVAL}(\cdot)$ and $\text{RANGE}(\cdot)$ problems for simple acyclic PRSs. Since $\text{EVAL}(\cdot)$ clearly reduces to $\text{RANGE}(\cdot)$ in this case, we conclude the following.

THEOREM 6.5. *For simple and acyclic PRSs and for all representations,*

1. $\text{POLY}(\cdot) \in$ EXPTIME*,*
2. $\text{RANGE}(\cdot), \text{EVAL}(\cdot)$ *are* PSPACE*-complete.*

6.3. Idempotent systems. Note that, if S is simple and deterministic, then $\text{POLY}(S_{\text{idem}})$ and $\text{POLY}[S_{\text{idem}}]$ are finite, and we obtain results analogous to Theorem 6.1.

THEOREM 6.6. *Let S be simple, deterministic, and acyclic. For all representations the following hold:*

1. $\text{POLY}(S_{\text{idem}}) \in$ P.
2. $\text{RANGE}(S_{\text{idem}}) \in$ NP*. In fact, there are systems S such that the problem* $\text{RANGE}(S_{\text{idem}})$ *is* NP*-complete.*
3. $\text{EVAL}(S_{\text{idem}}) \in \text{TC}^0$.

For the variable membership problems the following can be said.

THEOREM 6.7. *For idempotent, simple, deterministic, and acyclic PRSs and for all representations,* $\text{POLY}(\cdot), \text{RANGE}(\cdot), \text{EVAL}(\cdot) \in$ EXPTIME.

Proof. The proof follows by the trivial evaluation of the system. □

Lemma 5.13 shows the importance of the minimization and maximization operations in the case of idempotent systems.

THEOREM 6.8. *For idempotent, simple, deterministic, and acyclic replacement systems and for all representations,*

1. *the functions* $\min \text{EVAL}(\cdot)$ *and* $\min \text{EVAL}[\cdot]$ *are* #P*-complete under* $\leq^{\log}_{1\text{-T}}$*-reductions*;

2. *the functions* $\max \text{EVAL}(\cdot)$ *and* $\max \text{EVAL}[\cdot]$ *are* FP*-complete under* $\leq^{\text{p}}_{m}$.

Proof. The proof is an immediate consequence of Lemma 2.1, Theorem 2.4, and Lemma 5.13. □

For completeness, we note the following.

Remark 6.9. For simple deterministic and for simple acyclic PRSs and for all representations, the functions $\min \text{EVAL}(\cdot)$, $\min \text{EVAL}[\cdot]$, $\max \text{EVAL}(\cdot)$, $\max \text{EVAL}[\cdot]$ are FP-complete.

Proof. For simple deterministic S and $a \in \mathbb{N}^*$, obtain from Lemmas 5.7 and 6.1 the slp representation of the unique polynomial p in $\text{POLY}(S)$ (if there is one). Then compute $p(a)$ which hence is the minimum and maximum. This element can be computed as in Theorem 6.2, showing that the functions are in FP. Hardness follows immediately from Lemma 2.1.

For simple acyclic S, we evaluate S according to a topological order of G_S. The minimal element is obtained if for every variable x we consider only those rules with left-hand side x, whose right-hand side is minimal. Hence we essentially deal with a deterministic system, and the upper bound follows from the above. The case of maximization is treated similarly. Hardness follows from hardness of the problem in the deterministic case. □

7. Conclusion. It may be of interest to consider the complexity of the proof-circuit problem PC (see section 2) for circuits of restricted depth. The circuit constructed in the proof of Theorem 2.4 has depth 5. Simply merging the $\wedge$-gates on levels 3 and 4 makes it depth-4. Furthermore, it is easy to see that for depth $d+1$ circuits with an $\vee$ output-gate the problem is as hard as for depth-d circuits with an $\wedge$ output-gate. Next observe that the problem is easy (i.e., in FP) for depth-2 circuits with an $\wedge$ output-gate (and hence also for depth-3 circuits with an $\vee$ output-gate).

This means that the complexity of PC is not known only in the case of depth-3 circuits with an $\wedge$ output-gate (or, equivalently, for depth-4 circuits with an $\vee$ output-gate). It is not hard to see that this reduces to the case of four-level stratified circuits with

(i) level 1: 1 gate,
(ii) level 2: $\vee$ gates,
(iii) level 3: $\vee$ gates,
(iv) level 4: $\wedge$ gate (output-gate).

This problem in its turn is equivalent to the following problem: Given natural numbers $b_1, \ldots, b_m$ and the polynomial $f(x_1, \ldots, x_m) = \prod_{k=1,\ldots,n} \sum_{i=1,\ldots,m} a_{ki} \cdot x_i$ with $\text{a}_{ki} \in \{0, 1\}$, exploit distributivity to re-express this polynomial as a sum of monomials, and replace every x_i^r, $r > 1$, with x_i. Let $f^*(x_1, \ldots, x_m)$ be this new polynomial. Goal: Compute $f^*(b_1, \ldots, b_m)$. (The complexity is measured in the value—i.e., unary—of $n + b_1 + \cdots + b_m$.)

It is interesting to see that the problem of computing the permanent can be formulated in a way similar to the above problem, only substituting "replace every x_i^r, $r > 1$, with x_i" by "replace every x_i^r, $r \geq 1$, with 0." Nevertheless, it is not clear how to use this idea to reduce the problem of computing the permanent to the above problem.

Determining the complexity of the sets $\text{RANGE}(S), \text{RANGE}[S]$ for fixed S is often equivalent to determining the complexity of the range of a multivariate polynomial with nonnegative integer coefficients. While this is always in NP, we showed that

there is a four-variable polynomial of degree 6 whose range is NP-complete. Can this be improved?

From a complexity-theoretic point of view, further obvious open questions are whether the gaps between lower and upper bounds in Theorems 6.5 and 6.7 can be closed.

Many other questions about PRSs remain open. Returning to some of the problems posed in subsection 1.1, we did not study multivariate rules at all. Also, it may be worth examining if PRS families other than idempotent systems can be related to counting problems involving arithmetic or Boolean circuits.

Acknowledgments. We thank Sven Kosub (Würzburg) and Thomas Thierauf (Ulm) for helpful discussions. Also we are grateful to an anonymous referee for pointing out an oversight in Theorem 6.1 and for comments that helped to improve the presentation.

REFERENCES

[1] M. Agrawal, E. Allender, and S. Datta, *On TC^0, AC^0, and arithmetic circuits*, J. Comput. System Sci., 60 (2000), pp. 395–421.

[2] E. Allender, *Making computation count: Arithmetic circuits in the nineties*, SIGACT News, 28 (1998), pp. 2–15.

[3] E. Allender, A. Ambainis, D. M. Barrington, S. Datta, and H. LêThanh, *Bounded depth arithmetic circuits: Counting and closure*, in Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 1644, Springer-Verlag, Berlin, Heidelberg, 1999, pp. 149–158.

[4] A. Ambainis, D. Mix Barrington, and H. LêThanh, *On counting* AC^0 *circuits with negative constants*, in Proceedings of the 23rd Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 1450, Springer-Verlag, Berlin, 1998, pp. 409–417.

[5] L. Babai and L. Fortnow, *Arithmetization: A new method in structural complexity theory*, Comput. Complexity, 1 (1991), pp. 41–66.

[6] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer, *Nondeterministic* NC^1 *computation*, J. Comput. System Sci., 57 (1998), pp. 200–212.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[8] O. Goldreich, *Modern Cryptography, Probabilistic Proofs, and Pseudorandomness*, Algorithms Combin. 17, Springer-Verlag, Berlin, Heidelberg, 1999.

[9] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison–Wesley Series in Computer Science, Addison–Wesley, Reading, MA, 1979.

[10] O. Ibarra and S. Moran, *Probabilistic algorithms for deciding equivalence of straight-line programs*, J. ACM, 30 (1983), pp. 217–228.

[11] J. Jiao, *Some Questions Concerning Circuit Counting Classes and Other Low-level Complexity Classes*, Master's essay, Rutgers University, New Brunswick, NJ, 1992.

[12] E. Kaltofen, *Greatest common divisors of polynomials given by straight-line programs*, J. ACM, 35 (1988), pp. 231–264.

[13] K. J. Lange, *Unambiguity of circuits*, Theoret. Comput. Sci., 107 (1993), pp. 77–94.

[14] H. LêThanh, *Circuits arithmétiques de profondeur constante*, Ph.D. thesis, Université de Paris-Sud, Paris, France, 1998.

[15] Y. V. Matiyasevich, *Hilbert's Tenth Problem*, Found. Comput. Ser., MIT Press, Cambridge, MA, 1993.

[16] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[17] C. H. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994.

[18] J. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, J. ACM, 27 (1980), pp. 701–717.

[19] L. J. Stockmeyer and A. R. Meyer, *Word problems requiring exponential time*, in Proceedings of the 5th ACM Symposium on Theory of Computing, ACM Press, New York, 1973, pp. 1–9.

[20] I. H. Sudborough, *On the tape complexity of deterministic context-free languages*, J. ACM, 25 (1978), pp. 405–414.
[21] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, *Fast parallel computation of polynomials using few processors*, SIAM J. Comput., 12 (1983), pp. 641–644.
[22] L. G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput., 8 (1979), pp. 410–421.
[23] H. Venkateswaran, *Circuit definitions of nondeterministic complexity classes*, SIAM J. Comput., 21 (1992), pp. 655–670.
[24] H. Venkateswaran and M. Tompa, *A new pebble game that characterizes parallel complexity classes*, SIAM J. Comput., 18 (1989), pp. 533–549.
[25] V. Vinay, *Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits*, in Proceedings of the 6th IEEE Conference on Structure in Complexity Theory, IEEE Computer Society, Los Alamitos, CA, 1991, pp. 270–284.
[26] V. Vinay, *Semi-Unboundedness and Complexity Classes*, Ph.D. thesis, Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India, 1991.
[27] H. Vollmer, *Introduction to Circuit Complexity. A Uniform Approach*, Texts Theoret. Comput. Sci. EATCS Ser., Springer-Verlag, Berlin, 1999.
[28] K. W. Wagner, *The complexity of problems concerning graphs with regularities*, in Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 176, 1984, pp. 544–552. Tech. rep., Friedrich-Schiller-Universität Jena, 1984.
[29] K. W. Wagner, *The complexity of combinatorial problems with succinct input representation*, Acta Inform., 23 (1986), pp. 325–356.
[30] K. Yang, *Integer circuit evaluation is PSPACE-complete*, J. Comput. System Sci., 63 (2001), pp. 288–303.
[31] R. Zippel, *Probabilistic algorithms for sparse polynomials*, in Proceedings of EUROSAM 79, Lecture Notes in Comput. Sci. 72, Springer-Verlag, New York, 1979, pp. 216–226.